ADAPTIVE AUTOMATION DESIGN AND IMPLEMENTATION

DISSERTATION

Jason M. Bindewald, Capt, USAF

AFIT-ENG-DS-15-S-007

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-15-S-007

ADAPTIVE AUTOMATION DESIGN AND IMPLEMENTATION

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Jason M. Bindewald, B.A.C.S., M.S.C.O.

Capt, USAF

17 September 2015

AFIT-ENG-DS-15-S-007

ADAPTIVE AUTOMATION DESIGN AND IMPLEMENTATION

DISSERTATION

Jason M. Bindewald, B.A.C.S., M.S.C.O.
Capt, USAF

Committee Membership:

Gilbert L. Peterson, PhD
Chairman

Michael E. Miller, PhD
Member

LtCol Brent T. Langhals, PhD
Member

Adedji B. Badiru, PhD
Dean, Graduate School of Engineering and Management

AFIT-ENG-DS-15-S-007

Abstract

To increase human safety, reduce manpower costs, and increase human effectiveness; the Air Force is moving toward more autonomous systems, such as unmanned aerial vehicles. As more and better autonomy is added to these platforms, situations exist that still require human intervention. Adaptive automation is a research field that addresses the complex interactions within human-machine systems by enabling the overall system to adjust to changing environments by changing how the machine operates and/or interacts with the human on the fly. Within the adaptive automation research field, several theoretical aspects of adaptive automation systems have been described. This research contributes processes and insights for practitioners to move from a theoretical adaptive automation goal to a real-world system, answering the question, "How do we create a real-world adaptive automation system around a specific adaptive automation goal?" Answering this question produces three primary contributions: the Function to Task Design Process Model for designing adaptive automation in a human-machine system, a real-time player modeling framework for imitating a specific person's task performance, and the Adaptive Automation System Design Life Cycle for moving designed adaptive automation systems to implementation. We demonstrate these contributions through a real-world adaptive automation implementation.

Acknowledgements

I am indebted to my research advisor, Dr. Bert Peterson, for his guidance and patience. My research committee, Dr. Michael Miller and LtCol Brent Langhals, have been valuable mentors. Thanks to Dr. Kennard Laviers for implementing the *Space Navigator* system and Mr. Bryan Zake for updating the system for future experiments. I would not have been able to complete this research without the help of more than fifty volunteers who completed the experiments, some completing as many as four iterations.

Most critical to this work have been my wife and children. Their motivation is the fuel for any successes I may achieve.

Jason M. Bindewald

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

I. Introduction

According to Parasuraman *et al.* [4], automation is having "a computer carry out certain functions that the human operator would normally perform". Whether a task is automated or not is determined by which entity is performing a given task. There are many types of tasks. Consequently, automation can take several forms, as the type of automation is determined primarily by the type of task automated, which fall into four primary categories: sensors (acquiring information); computers (processing information); actuators (acting on the environment); and communicators (transferring processed data with humans) [5]. Since it can take many forms, *automation* will heretofore be defined as the performance of a human task by a non-living system.

In many cases the way an automation interacts with a human is set. However, there exist purposes for which adapting the automation can be useful. According to Merriam-Webster's Dictionary, the definition of *adapt* is"to make fit (as for a new use) often by modification" [6]. In this regard, adaptation can be useful in helping to not only move between the different types of automation, but also between different levels and stages of automation. The idea of developing systems that adapt automations on the fly was first proposed by Rouse [7] and was later developed further [8]. The terms *adaptive automation*, *adaptive systems*, and *dynamic systems* have all been used interchangeably to refer to these changing systems.

The specifics of what adaptive automation entails is cause for some debate. Much has been written on different types of automation. While some have focused adaptive automation only on systems where the machine adapts to the environment [9], others

have explained that humans can serve as the decision-maker in deciding when to adapt automation [10]. Some have defined adaptive automation as systems with different levels of automation and the ability to trigger changes between those levels [4], while others have focused on looking at the construction of the actual triggers as an adaptive automation [11]. For the purposes of this research the term *adaptive automation* refers to a human-machine system that dynamically adjusts some portion of the system's operation to the changing environment in which the overall human-machine system operates.

The motivation in researching adaptive automation stems from the need for automated systems that can account for irregularities within the operation of the natural world, and human beings in particular. Humans can behave unpredictably, and to a lesser extent the computer systems that automate tasks encounter unforeseen circumstances. As such, the effectiveness of automations can vary drastically depending on several variables. If the human operator within a partially automated system is suffering from a lack of sleep, he may not perform the task at the level for which the system was designed. Contrarily, a person who is extremely well-suited to a given automated tasks may become bored with a task that he would be able to perform more effectively and efficiently than the computer automating it. Therefore, the motivation for adaptive automation boils down to one of dealing with the variability of human and envrionment within the context of automated systems.

1.1 Adaptive Automation in Practice

In practice, adaptive automation research has provided tools for operation within many different fields. These areas range from real-world task performance to artificial control tasks designed as test-beds. The following systems were designed for other purposes, but have since been used for AA specific purposes.

- *aCAMS* [12, 13] - The Cabin Air Management System (CAMS) was developed to study the effects of sleep deprivation. It was later modified to support different levels of automation [14]. The Automation-Enhanced Cabin Air Management System (aCAMS) provides a simulated version of a spaceship's life support system, controlling five sub-systems: O_2, CO_2, pressure, temperature, and humidity. Since its creation, aCAMS has been used in several adaptive automation-based research studies [15–20].

- *ALOA* [21] - The Adaptive Levels of Autonomy (ALOA) environment was created as a testbed for the evaluation of adaptive automation schemes. The system was specifically designed around the application area of multiple unmanned aerial vehicles (UAVs), and the supervisory control thereof. The system allows for different levels of automation among four operator tasks: weapon release authorization, image analysis, allocation of UAV to mission objectives, and autorouting of UAVs. The system has subsequently been used in a succession of adaptive automation research efforts [22–25].

- *Communication Scheduler* [26] - The Communication Scheduler is a two-part adaptive system to allow for prioritization of communications for reducing cognitive strain on soldiers. This system consists of two parts: a cognitive state assessor and a communication scheduler. The cognitive state assessor allows the system to read in information about the subject, while the communications scheduler then uses this information to determine how to prioritize incoming communications. The research platform has allowed for further research into adaptive automation [9, 27].

- *MAT-B* [28] - The Multi-Attribute Task Battery (MAT-B) was developed as a benchmark set of tasks for the measurement of workload. MAT-B consists of

four tasks, representing the areas of: communications, resource management, monitoring, and tracking. The battery was designed to model tasks that may be performed by an aircraft crew. Although it was not designed specifically for adaptive automation, it has been used in several adaptive automation research efforts [29–32]. The Shared Attribute Task Better (SAT-B) is another testbed that is based on the MAT-B, but allows two human operators to interact on the tasks rather than just one [33].

- *MIX Testbed* [34] - Similar to many of the other testbeds, the Mixed Initiative Experimental (MIX) Testbed is an environment designed to investigate how unmanned systems are used and to capture information on automation of them. The testbed includes an unmanned vehicle simulator, which provides simulations of UAVs and UGVs. The Operator Control Unit (OCU) system provides a graphical user interface for interacting with UGVs. Auditory and visual monitoring tasks are available as well [35, 36].

- *MultiTask©* [37, 38] - The MultiTask©environment is essentially a radar monitoring task, wherein users are required to eliminate square targets moving to the center of a "radar scope" display. This must be done before the targets collide with each other or reach the center of the display, by selecting the target with a mouse or keyboard. Multitask is meant to roughly simulate a real-world radar or air traffic control monitoring task. Several follow-up studies have used the environment in further adaptive automation research [39–45].

- *Networked Fire Chief* [46] - The Networked Fire Chief is a fire fighting simulation designed to allow research into the psychological constraints involved in decision-making. The operator is presented with fires that pop up throughout an area of land, and he must process contributing factors and allocate resources

to put out the fires. This system has not yet been used in direct adaptive automation implementations, but has been discussed as a potential avenue for future adaptive automation research [47, 48].

- *N-DART* [49] - The Naval Dynamic Allocation Research Testbed (N-DART) was developed by Navy researchers to help overcome some of the problems that the MAT-B presents to Naval research. The system helps to automate the resource allocation stage of Naval Command and Control tasks. The overall goal of the system is to counter all incoming threats to a set of naval subjects controlled by the operator. The task is divided into two sub-tasks: a resource allocation task and a communications task [50, 51].

- *RESCHU* [52–56] - The Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles (RESCHU) Simulator is a test-bed wherein a single human operator is allowed to control a team of unmanned vehicles (UVs). These UVs consist of both unmanned aerial vehicles (UAVs) and unmanned underwater vehicles (UUVs). The UVs work together to perform a surveillance task, locating objects of interest. The simulation environment consists mainly of three sections: a map window, a payload window, and a status window. One of the main purposes of RESCHU is to force the operator to execute more than one task at the same time.

- *Robotic NCO* [57] - The Robotic Noncommissioned Officer (NCO) Program was developed to test the feasibility of adaptive automation in an environment where a single operator controls multiple robotic systems. The environment includes three main tasks: route-planning with unmanned ground vehicles (UGVs), target identification using UAV sensors, and multi-level communications. Further research efforts have made the capabilities of the environment

more robust [58, 59] and investigated adaptive automation further [60, 61].

- *SCARLETT* [62, 63] - The Supervisory Control and Response to Leaks: TARA at Tsukuba (SCARLETT) microworld was created as a central heating system control task. Two tasks are included in SCARLETT: controlling the temperature in an apartment complex central heating system and managing faults that arise within the central heating system. The types of faults that can occur include leaks, breaks, and accidents. Experiments used with SCARLETT have adaptively automated along the line of choice and/or performance of a given task within the system.

- *SIL* [64] - The Simulation Integration Laboratory (SIL) is an environment developed to allow for simulation of single operators controlling multiple UAVs and/or UGVs in scenarios. The SIL is composed of three main entities: a military entities simulator, a 3-D virtual world simulator, and a tactical control unit (TCU). The testbed allows for a few different monitoring tasks as well as a classification task [65].

- *Simulated UCAV Task* [66] - The simulated Uninhabited Combat Air Vehicle (UCAV) task requires the operator to monitor a set of UAVs as they flew on a combat mission. The operator was required to prioritize targets and call for weapons release on targets. Additionally, each vehicles condition required monitoring and the operator was required to select appropriate maintenance actions as needed. The UCAV task was designed to test a method for adaptively adjusting an automated aiding level according to cognitive workload levels [67].

- *SKEM* [68, 69] - The STEP compliant Knowledge Engine for Manufacturing (SKEM) represents a knowledge base schema for use across different layers of factory production. The system was designed to support the ability to create

adaptive controls for use in a factory production setting. This is done by providing a bridge between the management of the production processes and the execution of the production plans by the production environment.

- *Telerobot* [70] - Telerobot is a simulated robot to demonstrate how the 10 levels of automation model could be used in designing automations for telerobots. The specific task environment simulated was telerobot control within a simulated nuclear materials handling task. The goal of the task is to safely handle plutonium throughout the assigned task. Further research efforts expanded the telerobot environment into adaptive automation research [71].

Although there are several systems, there are a few key drawbacks to using them. Some environments provide tasks that are not very complex, such that a computational system can perform optimally in the environment and making human interaction pointless. Additionally, sometimes the task environmnets are extremely difficult to understand and therefore do not allow for experiment participants to quickly understand and perform within the environment. Lastly, these environments tend to lack a competitive nature or other inducements that will encourage participation apart from research perspectives.

1.2 Research Motivation

Adaptive automation is a domain of research specifically relevant to the U.S. Department of Defense (DoD) and Air Force (USAF). Organizations at both levels have created a research emphasis on autonomous systems, with a specific focus on adaptive automation in several domains. As practitioners aim to fulfill these research goals, several researchers have created AA systems, as outlined above. However, there is a lack of a codified formal AA system design and implementation methodology.

Air Force Relevance.

The Defense Science Board Task Force on the Role of Autonomy in DoD Systems released a report on 19 July 2012 [72], aiming to "assist the DoD in identifying new opportunities to more aggressively use autonomy in military missions, to anticipate vulnerabilities, and to make recommendations for overcoming operational difficulties and systemic barriers to realizing the full potential of autonomous systems." This report emphasizes the ever-increasing need for autonomous systems specifically within the DoD, but also within the larger world.

The report singles out the responsibility trade space, where a delegation of responsibilities between humans and machines occurs. For example, switching task responsibility in operating unmanned vehicles (UVs) provides a potential manpower savings, but making this switch shifts responsibility or risk for critical decisions to a machine. This is a problem because critical decisions are often very complex and cannot be optimally solved by a machine, especially when life-or-death consequences are involved. Therefore, "smartly" switching between human and machine operators within an environment is a burgeoning research area. The DoD's Unmanned Systems Roadmap [73] points out that operation of multiple UAVs by a single human would enable reallocation of current human practitioners.

Switching operational control occurs within multi-aircraft control (MAC). Within MAC, a single operator controls multiple UAVs requiring the use of human-machine interfaces that can switch between tasks. To achieve MAC, the handling of a task must be automated to an extent where the human-assigned tasks can be performed by a single human across multiple missions [74]. Even within this one area, there are multiple sub-tasks where task switching must be addressed.

The Office of the US Air Force Chief Scientist [75] also pointed out the need for automation research, specifically emphasizing that over the next twenty years there

must be a movement away from human-controlled task environments to autonomous systems. In reference to AA systems that deal with switching between human and machine throughout operation, the report states that "the Air Force, as one the greatest potential beneficiaries of such systems, must be a leader in developing the underlying science and technology principles." The need for adaptive automation spans a host of domains including cyberspace infrastructure, distributed networks, training environments, aircraft operation, surveillance, trust systems, navigation, space communications, and spectrum warfare.

A common test-bed to investigate AA research priorities provides a motivation for investigating AA system development methodologies. The *Space Navigator* test-bed was created to fulfill this purpose, serving as the test-bed for the automation and human-machine systems designed for this research. *Space Navigator* provides a research environment that involves a complex dynamically changing environment, while allowing for users to understand and grasp the actions of the environment quickly. Additionally, creating a game-based test environment helped to attract experimental participants.

Space Navigator is a tablet computer based route creation game, shown in Figure 1. The game operates as follows. Four stationary planets are present on the screen. Each planet is one of four colors: red, green, blue, or yellow. Spaceships appear randomly from the sides of the screen. Each spaceship is red, green, blue, or yellow. The players must direct each spaceship to the destination planet of the same color by drawing trajectory lines on the game screen using their finger. Spaceship then follow these lines at a constant rate. If desired, trajectories may be re-drawn, to avoid a collision and account for dynamic changes (e.g. appearance of ships and bonuses, or movement of NFZs).

For the purposes of the experiments completed during this research, game set-

tings are applied as follows. The score increases by successfully landing a ship at its destination planet (+100 points) and by picking up small bonuses (+50 points) that randomly appear throughout the play area. The score decrements when spaceships collide, and all spaceships involved in the collision are destroyed (-100 points/spaceship lost). For each second a spaceship traverses one of several "no-fly zones" (NFZs) which, every fifteen seconds, move randomly throughout the play area; the score decrements (-10 points/second). Each game ends after five minutes.

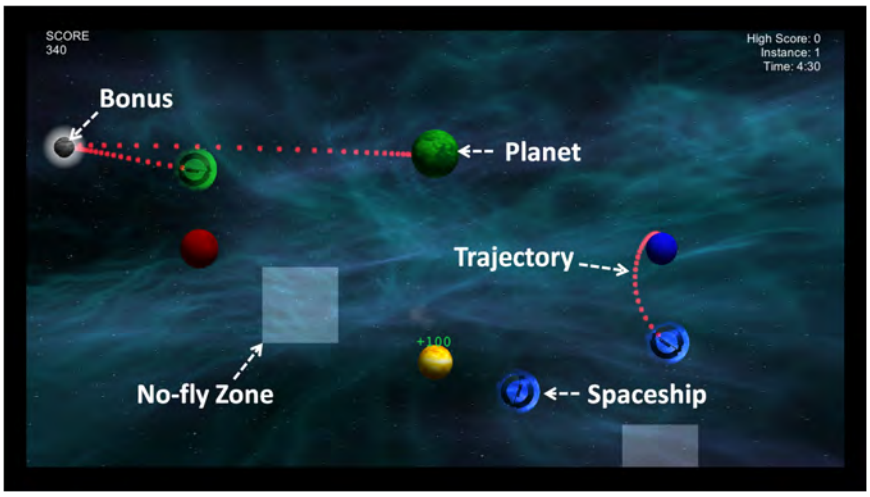

Figure 1. Screen capture from a game of Space Navigator, pointing out spaceships, planets, trajectories, bonuses and no-fly zones.

System Design and Implementation.

Many of the formal AA frameworks [4, 76] and taxonomies [3] provide insight for the developer into how to more effectively create an adaptive automation system, but they address the design and implementation aspects of the process somewhat indirectly. Although there are informal references to system design throughout the literature, many of these only provide insights into a specific aspect of the design. It has been suggested that specific adaptive automation design guidelines could provide a specific place for further research, as there is a dearth of formal literature addressing

the subject [41].

One methodology proposed by Parasuraman [77] includes a six step process designed around the types and levels of automation. The six steps include: Identify types of automation; identify level of automation; apply primary criteria (mental workload, situational awareness, complacency, skill degradation) to each level/type pair identified; set initial specification of types and upper bound of levels of automation; apply secondary evaluative criteria; and final specification of types and levels of automation. Another methodology specifically designed to assist in the early stages of automation interface design, developed by de Visser *et al.* [78], consists of five steps: collect observational data of a system; conduct task analyses; construct a quantitative model; create preliminary design; and validate design. This lays out a design methodology, but more could be done to formalize these methods. A third AA design methodology deals only with AA triggers to determine when to turn an AA on and off [11]. None of these design methodologies addresses the entirety of an AA system to include not only the human and machine elements of the system, but also how and when they interact.

In addition to a methodology for design of systems there is a further lack of a codified AA system design and implementation life-cycle. Several considerations must be made to ensure successful research. Haarman *et al.* [79] suggest that realistic implementations for AA are extremely important in complex environments, such as the ones targeted by DoD and USAF research. Several of the AA systems mentioned above are created in different manners, and may not properly address complex environments that they are meant to represent. To allow for the design of AA in increasingly complex environments, a codified system design and implementation concept would help bring the larger AA community together in designing more effective AA systems that can rely on similar practices to report problems and insights

into real-world implementations.

1.3 Research Purpose and Contributions

Automated systems are placed within a human process to achieve some *over-arching goal*. The resulting human-machine system aims to achieve this overall goal, such as having a pilot (human) and an autopilot system (machine) working together to successfully complete a UAV flying mission. Adding adaptive automation to the human-machine system requires the definition of a specific adaptive automation goal. An *adaptive automation goal* explains why the human and machine are interacting within the system in an adaptive manner. An AA goal should not define how the human and machine will interact, but rather explain how adding adaptivity will help the system achieve the over-arching system goal. This dissertation aims to show how practitioners can move from a theoretical AA goal to a real-world adaptive automation system, answering the question "How do we design and implement a real-world adaptive automation system around a specific adaptive automation goal?"

The process of answering the research question led to the three fundamental contributions of this dissertation. First, the Function to Task Design Process Model outlined in [1] provides a theoretical framework for pinpointing good locations for AA within a human-machine system. Second, implementing the specific AA goal chosen for this research (performing a sub-task similarly to the user) into an actual automated system led to a novel clustering-based real-time player modeling framework. Third, the process of implementing the automation in an adaptive manner led to the creation of the Adaptive Automation System Design Life Cycle (AASDLC). Each of the three major contributions additionally led to other novel contributions and insights to their respective fields.

The Function-to-Task Design Process Model.

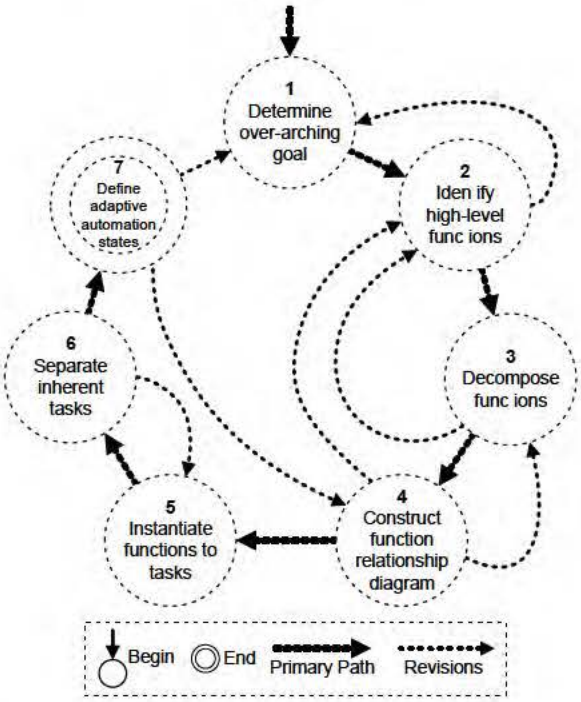

Figure 2. The Function-to-Task Design Process Model as presented in [1].

The Function to Task Design Process Model (FTTDPM) for designing systems with built in AA is a seven step process shown in Figure 2 [1]. In addition to the overall design process, the FTTDPM provides three unique contributions to the field of adaptive automation system design. First, we explain the process of function instantiation and introduce the concept of inherent tasks that arise through the instantiation process. Next, we propose a set of three progressively additive visual diagrams enabling better human and machine function instantiation. Finally, we develop five analysis tools for isolating effective AA points within a human-machine system.

A *function* is an action that an element of a system performs to accomplish the desired goals or to provide the desired capability; it is not allocated to an entity. A *task* is a function the performance of which has been allocated to a specific performing entity (human or machine)–a task is an instantiated function. Instantiating a function

as either a human or machine task will elicit a single explicit task and some number of inherent tasks. An *explicit task* is directly indicated by a previously defined function–there is a one-to-one mapping between function and task. *Inherent tasks* arise as a direct result of function instantiation–as a result of assigning performance of a function to a specific entity, specific functions that were not previously required now arise. An inherent task is one that is not necessarily needed by the function, but becomes necessary for the performing entity once the function is instantiated as a human or machine task. One of the most important inherent tasks involves passing along contextual information from one entity to the other, to ensure a proper decision-making environment. For example, an inherent task when passing pilot control of a UAV from an auto-pilot back to the pilot is that the machine must alert the human that control is about to pass to them.

The FTTDPM uses a set of three visual diagrams coupled with the inherent task concept to identify good locations for adaptive automation within a system. The Function Relationship Diagrams (FRD), Task Relationship Diagrams (TRD), and TRD with automation added (Auto-TRD) form a set of additive diagrams that represent a human-machine system graphically, specifically designed for depiction of systems with AA. The FRD, created in step four of Figure 2, is the basic representation of the system design with all of the functionalities the system will have to perform identified with connecting branches. The FRD is developed into the TRD during steps five and six in the process and accounts for both the explicit and inherent tasks that arise during task instantiation. The resulting diagram shows a system that represents not only the human-machine system's functions and allocated entities (human or machine), but also represents the inherent tasks associated with switching back and forth between human and machine task instantiation. Step seven of the FTTDPM further develops the TRD into the Auto-TRD by adding adaptive

automation nodes to the design, depicting locations in the design where control can switch freely between human and machine control.

Step seven in Figure 2 relies on five analysis tools used in conjunction with the TRD to help identify locations where the human-machine system would benefit from AA. These five tools include analyzing the number of possible states arising from the selection of a specific AA node location, identifying the number of different entity task handoffs picking a specific location for AA would create, determining where clusters of functionality are located within the TRD, counting the number of branches incident into and out of an AA node based on the location chosen, and comparing the increase in inherent task load from different AA location choices. These five tools, used in conjunction, help identify areas where adding AA would be easier within the confines of the currently designed TRD.

Clustering-Based Real-Time Player Modeling.

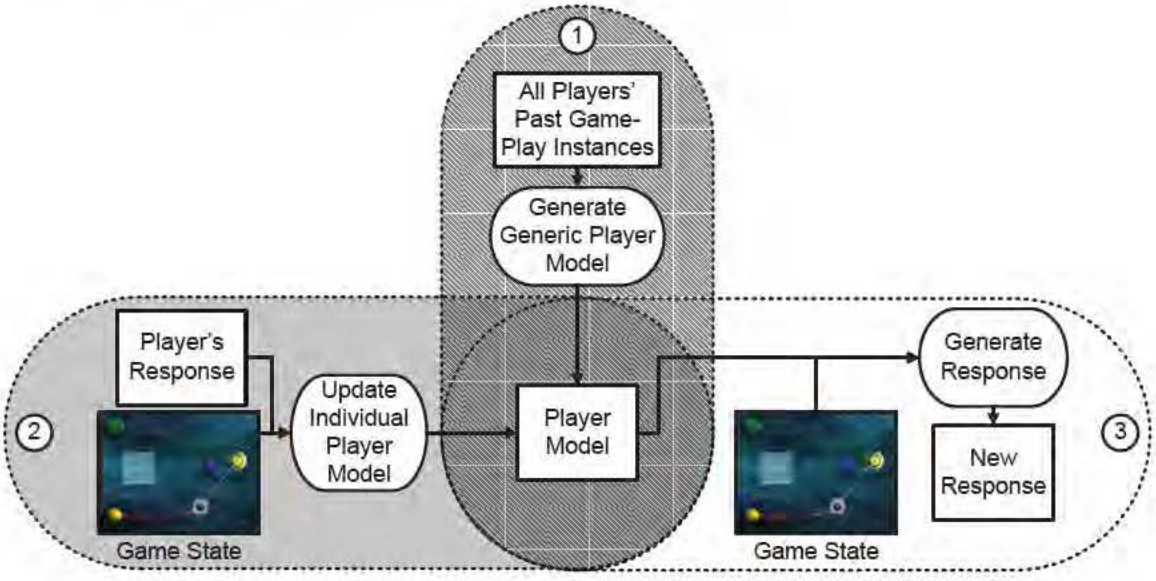

Figure 3. A real-time updating individual player modeling paradigm.

To investigate how an AA system is designed and implemented, an AA goal was

chosen. While most AA research efforts [76, 80] have focused on how the manner in which the automated portion of the AA performs its assigned task affects the performance of the human portion of the AA system, the focus we chose investigates how the manner in which an AA automates a task affects the overall human-machine team's performance. Specifically investigating how the similarity or dissimilarity of an automated aids task performance to that of the current operator affects the overall human-machine teams performance.

The clustering-based real-time player modeling methodology shown in Figure 3 was created to achieve the AA goal within the automated portion of the AA system design. The methodology presents a real-time individual player modeling system that enables an automation to perform response actions in its given environment that are similar to those that an individual player would have performed in a similar situation. The player modeling technique presented involves three major phases that can occur independently of each other or may overlap in a real-time system: (1) create a generic player model, (2) update the individual player model, and (3) generate a response using the player model. In addition to the process itself, the player modeling methodology provides three key contributions to the research area: the player model updates in real-time, it learns player tendency quickly, and it provides practitioners valuable insights into how the player interacts with the environment.

The player modeling system automatically updates in real-time by building individual player models from a generic player model. A generic player model is created through an agglomerative clustering of all state-response pairs by state and by response. Mapping the state clusters to response clusters and assigning each mapping a probability based on past game-play generates a player model. The player model is then made more generic by pruning state and response clusters that are unlikely to show up in most cases. This player model then updates during system operation by

mapping real-world states to the player's provided responses. These updates shape the generic model over time to better represent the specific player's game-play habits.

When applied in the *Space Navigator* environment, the player modeling methodology was able to obtain a statistically significant differentiation between players over five, five-minute games. This result is useful in that the system is able to distinguish between players in a relatively short amount of time. The improvements are gained by utilizing the generic model as a baseline for learning human game-play and then coupling that with a player model update algorithm that weights states differently based on the amount of utility assigned to the individual states. The update increment for each state-response pair is based on three traits of the clusters: cluster population, cluster mapping variance, and previous modeling utility.

The final individual player models provide meaningful insights when compared against the generic player model. We take the amount of change in the player model in comparison to the generic model and use that information to create a player model learning value. Analyzing the learning values for different state clusters in conjunction with the make-up of the attributes in each cluster's state representation helps the practitioner determine how different game attributes influence player behavior in the environment. Then, the correlation between player model learning values and the state representations with which they align helps the designer understand what sets apart individual players. These insights can then be leveraged to improve game design further.

An Adaptive Automation System Development Life-Cycle.

The final major contribution of this research involves moving from an adaptive automation design coupled with an automated system to a real-world adaptive automation system implementation. There have been plenty of adaptive automation

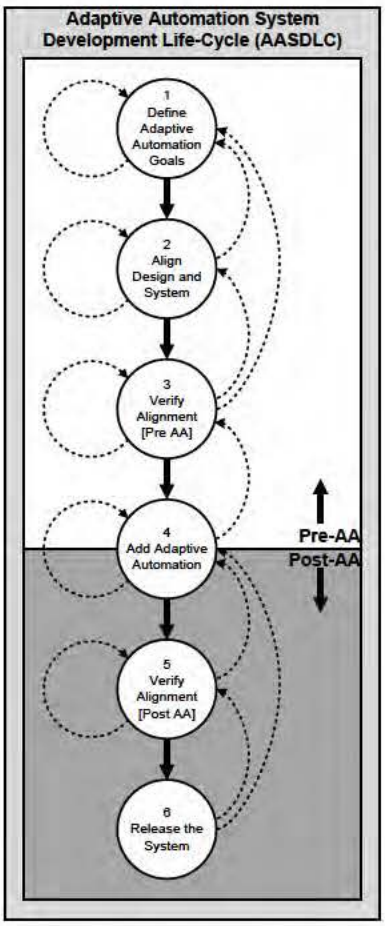

Figure 4. The adaptive automation system design life cycle.

taxonomies [3, 4, 76] and even an adaptive automation design methodology created for this research [81], but AA research lacks a start-to-finish system design life-cycle. Conversely, there exist several systems design life-cycles [82–86], but systems design research has not developed a life-cycle to address the idiosyncrasies of AA specific design and implementation. The Adaptive Automation System Development Life Cycle (AASDLC) is a start-to-finish development life-cycle shown in Figure 4. In conjunction with the major contribution of the life-cycle itself, the AASDLC contributes in three ways to the adaptive automation and system design communities: (1) incorporating the AA-centered design principles of the FTTDPM, (2) creating a new user feedback spectrum, and (3) developing a novel model for AA triggers.

The AASDLC is a six phase development life-cycle, through which the progression is not always linear. The life-cycle consists of two distinct phases, pre- and post-AA, and involves a series of cycles between aligning design with the real-world system and verifying the alignment through user testing. Seeking user feedback on design alignment early in the life-cycle allows the system designer to ensure that the system is accurately represented to ensure the AA added to the system is situated within the system well. Gaining user feedback after the addition of AA ensures that the real-world system aligns with any key elements of the design.

The system design aspects of the AASDLC rely on the principles of the FTTDPM. The TRD forms the basis for alignment between the design and the real-world system before AA is added. Then the five tools for determining where to place AA within the system are used to create an adaptive element within the system that aligns with the AA goal. After AA is added to the system, the Auto-TRD forms the basis for follow-up alignment between the design and system.

The AASDLC presents a unique model for developing user feedback. The user feedback design methodology works along two spectrums of feedback: qualitative vs. quantitative and directed vs. undirected. The feedback gained from user-testing is useful beforehand to ensure that the TRD represents the system and that the design has not overly-influenced by the practitioner's biases. The feedback devices provide useful information to the post-AA system by allowing the designer to verify that the AA acts in the intended manner and achieves the AA goal, and in turn contributes to the over-arching system goal.

The last contribution of the AASDLC is an innovative way to classify AA triggers that couples Feigh *et al.*'s [3] AA trigger type taxonomy with a trigger mode. Trigger modes include discrete, continuous, or complex triggers. These modes allow the designer to show the many ways that AA can adapt the system to a dynamic

environment. By modeling triggers in this fashion, insights from other fields can help to design more meaningful and capable adaptations.

1.4 Organization

The remainder of this dissertation is arranged as follows. Chapter II shows the Function-to-Task Design Process Model as published in [1]. Chapter III presents the clustering-based real-time player modeling methodology as discussed in [87]. Chapter IV provides an overview of the Adaptive Automation System Development Life-Cycle as presented in [81]. Chapter V presents a discussion of the overall research findings and suggests possible avenues for future work

II. Function-to-Task Process Model

This research begins to answer the research question (How do we design and implement a real-world adaptive automation system around a specific adaptive automation goal?) by focusing on the first portion of the question: How do we design adaptive automation systems? To answer this question, designers need a better understanding of how tasks are structured within the environment to which they are adding adaptive automation. Decomposing the functionalities of an environment and allocating those functions to performing entities (human or machine) allows sub-sets of tasks advantageous to adaptive automation to naturally emerge.

This chapter discusses how designers can take advantage of the decomposition and allocation processes to design effective adaptive automations. The Function to Task Design Process Model is the culmination of these efforts and is presented here as a minor adaptation to work published in the *International Journal of Human-Computer Studies*,[1] with information presented in previous chapters of this dissertation removed.

2.1 Introduction and Definitions

Consumer, commercial, and government systems increasingly apply automation, particularly in systems which involve time critical decisions and actions. These systems include manufacturing plant process control [68, 69, 88], aircrew and air traffic control [89], and remotely piloted or controlled vehicles [25, 61, 90]. Automation can improve the performance of systems without increasing manpower requirements by allocating routine tasks to automated aids, improving safety through the use of automated monitoring aids, and reducing the overall cost or improving productivity

[1] Bindewald, J. M., Miller, M. E., and Peterson, G. L. A function-to-task process model for adaptive automation system design. *International Journal of Human-Computer Studies 72*, 12 (2014), 822–834.

of systems [8]. Additionally, automation can permit removal of the operator from particularly undesirable or dangerous environments [91], increasing the safety and reducing stressors placed upon the operator.

Unfortunately, automation system designers have limited ability to project future events, and are often unable to adapt when unforeseen circumstances occur. As such, utilization of a human operator who can adapt to these unforeseen circumstances to provide system resilience is desirable [92]. With the inclusion of a human operator, other problems arise. Some include over-reliance on automation [93], placing inappropriate levels of trust in the automation [94–96], or losing situation awareness to preclude appropriate recovery from automation failures [93]. Further, as operators are not performing active control of the system, they may not practice the knowledge necessary to operate the system and can suffer from skill atrophy [97]. As a result, practitioners developed adaptive automation systems to maintain user engagement, without overloading operators [7].

Automation is the capability "to have a computer carry out certain functions that the human operator would normally perform" [4]. Knowing which entity will perform a given task helps determine whether to automate a task or not. There are many types of tasks, and consequently, several forms of automation. The categories of automation can include, "the mechanization and integration of the sensing of environmental variables; data processing and decision making; mechanical action; and 'information action' by communication of processed information to people" [5].

Since, Rouse proposed a dynamic approach to automated decision-making [7, 8], the field has adopted the terms *adaptive automation* and *adaptive systems* to define the idea of an automated system that can adapt to a changing environment. Within research, the definition of adaptive automation has been subject to debate. Most authors would agree that levels or types of automation change in an adaptive system.

For example, [9] define adaptive systems as those "allowing the system to invoke varying levels of automation support in real time during task execution, often on the basis of its assessment of the current context...invoking them only as needed". This view of adaptive automation places the onus of determining the current automation state on the system. However, others have shown that even the determination of who 'adapts' the system (e.g., the system, the operator, etc.) can fall on a sliding scale [90].

Within the current context, a *system* is a combination of hardware, software, and human operators that work together to accomplish one or more goals. As a focus of the paper is system design, the term *machine* refers to the combination of all hardware and software within the system with which the human operator interacts.

Although the terms *function* and *task* are sometimes applied interchangeably [98], clear differentiation of these terms leads to a better understanding of the proposed process model. Here, we define a *function* as an action that an element of a system performs to accomplish the desired goals or to provide the desired capability. A function is delineated from a task as the function is not allocated to an entity. A *task* is a function allocated to a specific entity, and represents the actions necessary for the entity to perform the function.

A task's allocation can be either explicit or inherent. An *explicit* task is one that is directly indicated by a previously defined function. Alternatively, an *inherent* task arises only once a function is allocated to a specific entity. An inherent task is not required by the function, but is necessary to enable the allocated entity to perform the function. For example, the system might require an operator to make a selection, requiring an explicit action. However, to make this selection, the operator will need to gather appropriate information from the system or environment and make decisions, each of which are inherent tasks. *Task load* then describes the number and difficulty

of tasks assigned to human operators, to which they must respond.

Workload refers to the impact of the task demand placed upon the operator's mental or physical resources. The variability in the task load imposed upon an operator (and the workload the operator experiences) originates from a number of sources. In addition to the variance of performance due to explicitly defined workload, the performance of the human operator may vary due to individual factors such as fatigue, stress level, motivation, and training level [99, 100].

This research presents a function-to-task design process model to aid the conceptual design of adaptive automation systems. The function-to-task design process model creates a set of visual diagrams enabling designers to better allocate tasks between human and machine. This is achieved through a set of five analysis tools allowing designers to identify points within a function network where the transitions between human and machine entities can facilitate adaptive automation. This paper proceeds as follows. Section 2.2 reviews the design processes currently in place for adaptive automation systems. Section 2.3 presents the function-to-task design process model. Section 2.4 illustrates the function-to-task design process model through a system design iteration. Section 2.5 presents conclusions summarizing the information presented.

2.2 Designing Adaptive Automation Systems

Discussions on the design of manned systems as a tool to aid allocation of functions or tasks between a human operator and a machine often cite Fitts' List [101] of tasks that machines tend to perform "better" than humans and those that humans perform "better" than machines. Fitts et al. discussed tasking the machine to perform routine tasks that require high speed and force, computational power, short-term storage, or simultaneous activities; and further propose leveraging the human's flexibility,

judgment, selective recall, and inductive reasoning to improve system robustness to unforeseen circumstances. They also acknowledge the limitation of humans to correctly employ these capabilities when overloaded due to excessive task demands or to maintain alertness and employ these capabilities when not actively participating in system control.

One may consider the allocation of functions between man and machine within a system as a multi-objective optimization, wherein designers optimize some combination of performance, safety, and robustness as a function of the tasks allocated to each component. The limitations of system and human capability shape this optimization, with a significant component of human capability quantified in terms of human workload. Adaptive automation system design assumes that the number and difficulty of tasks performed will vary over time, and the tasks allocated to the human or machine need to vary to provide the human operator with an appropriate workload.

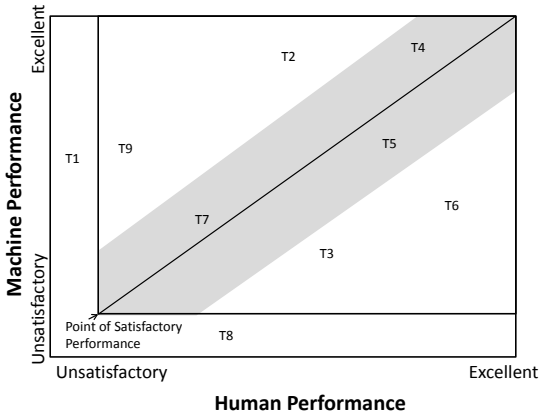

Figure 5. Diagram for task allocation in adaptive automation, adapted from [2].

Figure 5 illustrates this concept, which depicts a two-dimensional space which arranges tasks, T1-T9, based on how well a human operator or the machine can perform them under reasonable task load. As shown, performance by either system can range from unsatisfactory through excellent [2]. We should allocate tasks, such

as T1 or T8–which one entity (human or machine) can perform more satisfactorily–to the better performing entity. However, any task that either entity can perform beyond the point of satisfactory performance, we can reasonably allocate to either human or machine.

If there was no constraint on resources, one could maximize performance of the overall system by allocating tasks below the 45 degree line to the human and tasks above to the machine. However, resource constraints force a shift in the location of this line. For instance, assuming workload limits on human performance and unbounded machine resources might induce the designer to shift the dividing line lower in the plot, decreasing human workload and allocating additional tasks to the machine. On the other hand, if users' performances improve by increasing their engagement with the system, raising the dividing line allocates more tasks to the human. Therefore, adaptive automation effectively requires the system to permit this allocation line to shift up and down within this plot, allocating fewer or greater numbers of tasks to the human operator.

Automation Taxonomies.

Taxonomies for adaptive automation have been proposed to accommodate the complex design space present in adaptive automation systems. [3] indicate that modifying the allocation of tasks among humans or machines can affect operator workload. However, modification of task scheduling, interaction required between the operator and other system elements, or the content of any interaction can also affect operator workload. Although not explicitly captured, these modifications may involve systems with multiple machines or multiple humans [102]. Considering the interaction between an individual operator and a machine, [4] proposed a model for describing levels of automation that builds upon the work of [103] to discern between the types

and levels of automation. The model delineates the types of tasks performed based on the four-stages of human information processing: sensory processing, perception/-working memory, decision making, and response selection. Within these four stages, they expand upon the ideas of Sheridan and Verplank and codify them further into a 10-point scale describing the levels of automation, ranging from "1. The computer offers no assistance; human must take all decisions and actions" all the way to "10. The computer decides everything, acts autonomously, ignoring the human."

[76] proposed four core human functions that a system could automate independently of one another, including: monitoring, generating alternatives, selecting alternatives, and implementing the selected alternative. This framework assigns each of these four tasks to either the human or machine (or both in some cases) and enumerates the level of automation between fully autonomous and fully human-implemented, providing a two-dimensional space over which to define automation. Each of these classification schemes permits the differentiation between intermediate levels of automation, explicitly defining which human task a given level automates. Each model aids the creation and classification of automation states for tasks the human or machine can perform, helping the system designer determine "what" to automate and "to what extent"(i.e. level of automation). Although designers can apply "level of automation" models to any system employing automation, they are important in systems employing adaptive automation as they permit the designer to determine what part of and how to automate a task so that changes in automation level can be clearly described.

Although the adaptive automation taxonomy [3] proposes does not fully overlap the automation taxonomies provided by either [4] or [76], the taxonomies are not independent of one another. Feigh et al. uniquely highlight the fact that not all tasks are time critical, and systems can reprioritize them during periods of peak workload.

They also discuss the allocation of tasks between humans and machines–alluding to various levels for automation of tasks that include selection or implementation of alternatives. Additionally, they contend that automation of generating alternatives and monitoring requires automatically generated information displayed to the human operator, forcing a change in the interaction and content of interaction. Each of these methods, therefore provides a different way to classify and consider the effect of changes in autonomy on operator workload.

Human-Machine Interaction.

The need to provide effective communication between the human and machine impedes human interaction with automated systems. In some cases–such as flight control automation–the design of this interaction can have life-or-death consequences [41,104]. Unfortunately, this interaction can become increasingly complex in systems employing adaptive automation. William Rouse's analysis of human-machine interaction within a dynamic system is a seminal article in this field [8]. Rouse shows the different forms of communication with the system as a set of five interaction loops. The first two loops, in which it is possible that no communication is required, represented manual control and completely automated control. In the third loop, wherein he coins the term *overt communication*, the human and machine operators of a system directly communicate information about their tasks. The human operator must take explicit actions to control the machine, and the machine must explicitly provide information. The human operator must consciously read, listen to, or otherwise receive this information. The last two loops represent more subtle communication which typically occurs among humans; *covert communication*, with the fourth loop representing covert human to machine communication and the fifth covert machine-to-human communication. Information communicated indirectly–which might include

state information–characterizes covert communication. The timeliness of a response from a teammate, where hesitancy in response signals uncertainty and fast authoritative response indicates certainty, provides an example of covert communication.

Unfortunately, communication errors occur between human operators and machines as the machine can fail to communicate critical state information, the information leading to the selection of a critical state, or less direct information, such as the certainty of this information [104]. Recent research focuses on improving covert communication from the human to the machine through the use of psychophysiological measures, such as electroencephalography (EEG), electrocardiography (ECG), electrodermal activity (EDA), electromyography (EMG) [9,79,105] or behavioral measures, such as eye gaze patterns. Such measures have the "potential to yield real-time estimates of mental state" [105], thus allowing the machine to gain information regarding the state of the human operator.

The infeasibility of communicating all automated tasks from a machine to a human aside, the human in an automated system requires enough information to permit appropriate situation awareness. Since the human operator assumes control in the event of a mishap or in order to make a critical decision, the human needs an understanding of the current system and environment state. Several research efforts devote effort toward finding an appropriate balance between providing enough information for situation awareness and overloading the human operator with information [5, 41, 76, 106–108]. Further, all communication will affect the user's workload, potentially resulting in overload conditions. However, the relationship between how humans attend to, receive, process, and act upon information creates complexity, and the interaction influences the human operator's perceived workload [106].

The manner in which feedback is given influences the resulting system. For example, Manzey et al. demonstrate that users are much more likely to develop a

proper level of trust with a system when the system gives them negative feedback loops rather than positive ones [107]. Further issues, such as how to design a system to manage interruptions in a socially acceptable manner and analyzing the positive and negative consequences of automating the interruption management task [9], are important. The idea of etiquette flows naturally into the concept of trust, directly impacting the human operator's trust of the system.

While the design of the human-machine interface can be complex, this interface requires an understanding of the information that the human and the machine must communicate to facilitate task completion. The importance of this information necessitates its presentation in a way that does not overload the operator and recognizes the fact that the human operator will not necessarily receive all information the system provides.

User-Centered Design and Task Analysis.

User-Centered design [109] has evolved to aid the design of systems including a significant user interaction component. This process often involves the steps of: 1) establishing a vision for the system, including an initial system concept and business objectives; 2) analyzing requirements and user needs to understand how users perform the tasks within the boundary of the system concept and the context of use; 3) designing for usability through conceptual and detailed interaction design, including prototyping; 4) evaluating the system, which can involve early focused deployment and evaluation of the system; and 5) applying learnings from the evaluation to provide feedback and improvements to the overall system design [110]. Each of the steps in this process, as well as the overall process, are conducted iteratively until a desired level of usability is attained. During the development of complex systems, this process can involve individuals from numerous disciplines, to include systems engineering, human

factors, software design, human-computer interface design, and information system management, all of whom can apply different design and evaluation techniques during the vision development, analysis, and design phases.

Tools for capturing the requirements for a design within the systems engineering community include forms of the structured analysis and design technique (SADT) [111], as well as the Systems Modeling Language (SysML) [112]. SADT primarily focuses on the documentation and decomposition of the process to be employed by the machine during design. SysML was developed from the Unified Modeling Language (UML) [113], which was originally developed for the design of software systems. SysML includes a number of tools to capture and communicate the vision for the system, analyze requirements, develop conceptual and eventually detailed designs, and to associate test procedures and outcomes, permitting verification and validation of the system requirements. These tools can be applied in either a descriptive fashion, describing an existing system when analyzing requirements, or in a prescriptive fashion, documenting vision, requirements, and design of the system under design. While these tools focus primarily upon design of the machine, certain tools, including use case and activity diagrams can capture human interaction with the system.

Within the human factors community, Hierarchical Task Analysis (HTA) [114,115] is commonly applied to systematically capture and decompose human activities to describe processes that are commonly applied by the human. Information from these analyses can be depicted in a number of forms, one method of particular interest is the Operational Sequence Diagram (OSD) [116]. The OSD captures the flow of information between a human and machine, indicating the timing, direction and modality of information flow during each exchange between a human and the machine. More recently other task modeling languages have been developed. These tools include the ability to model tasks performed by groups of individuals, for example Groupware

Task Analysis (GTA) [117], as well as methods to assess differences between human and machine knowledge structures, e.g., Task Knowledge Structures (TKS) [118]. UML or SysML tools, can be applied to model the results of a task analysis. For example, use cases may represent an informal task analysis structure [119] and activity diagrams can capture tasks as well as information flow between entities [120].

Besides these tools, task description languages extend descriptive task analysis to provide prescriptive tools useful in specifying the design of the human interface. The Goals, Operators, Methods, and Selection Rules (GOMS) method [121] attempts to define the system through a set of goals, decomposing goals, determining how users perform tasks to achieve these goals and assessing the method of interaction on user performance. Other tools, such as DIANE+ [122] are useful during specification and user interface design [123]. ConcurTaskTrees (CTT) [124] uses a hierarchical task structure with a focus on defining a number of different types of temporal relationships between tasks to aid the design of the user interface.

These tools can be geared towards certain application environments. For example, AMBOSS [125] is primarily relevant to understanding human error in safety critical systems. Similarly the Functional Resonance Analysis Method (FRAM) [126] is primarily concerned with utilizing performance variability as it pertains to achieving desirable (or dampening undesirable) outcomes. Tools have also been developed in the management information field for capturing and designing workflows in complex business processes. Examples of these workflow languages are Yet Another Workflow Language (YAWL) [127] and Web Services Business Process Execution Language (WS-BPEL) [128].

In the context of the current research, a task model of particular interest is HAMSTERS and recent extensions to this model [129,130]. Similar to CTT, HAMSTERS provides notation for indicating machine tasks separately from human tasks when

decomposing a goal or high level function (i.e., abstract task in the HAMSTERS nomenclature). Each of these modeling languages additionally provide notation for interactive tasks, tasks performed by the human or system to facilitate interaction between the two entities. By creating different hierarchical networks with different tasks allocated to human or machine, these methods can be used to assess various static automation strategies. Further, HAMSTERS has recently been extended to permit the modeling of not only the hierarchical task relationship but the temporal sequencing of these tasks.

Existing task modeling tools are primarily designed for systems with static rather than adaptive automation. As a result, many of these tools do not provide tools to move across the function-task distinction described previously. In their designed context, these tools and methods typically do not explicitly provide ways to move from functions with an unassigned operational entity (e.g., machine or human) to tasks with an operational entity assigned. Therefore, a process model with a task model is proposed that aids the designer when determining functions to statically allocate to humans and machines, as well as functions to dynamically allocate between the human and machine.

2.3 Function-to-Task Design Process Model

This section presents the proposed process model for allocating functions to entities (e.g., human or machine) that leads to adaptive automation allocations. Figure 6 graphically depicts the function-to-task design process model. The function-to-task model usually proceeds in a linear fashion, as indicated by the solid bold arrows in Figure 6. In some cases, completing steps in the process will force the design back to a previous step for revision; likely locations for revision steps are indicated by the dashed arrows in Figure 6. It is likely that a proper decomposition of the system will

not occur on the first attempt and, therefore, this process can become iterative.

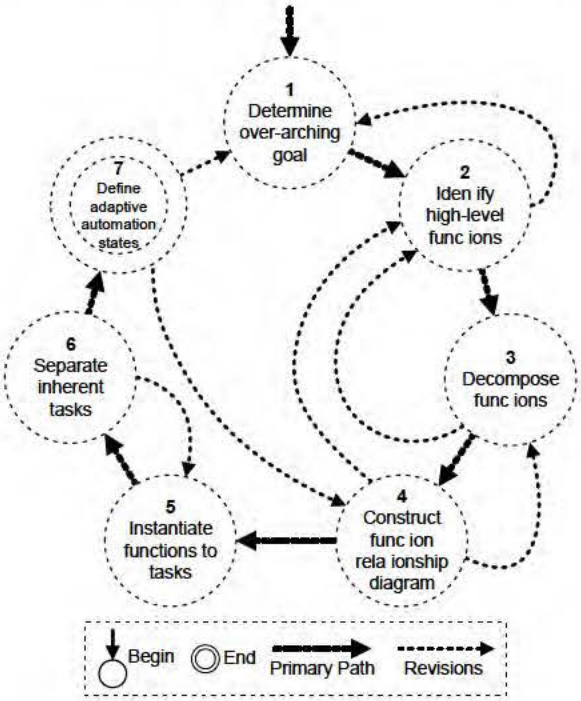

Figure 6. The Function-to-Task Design Process Model, depicting the developmental flow and typical revision loops necessary for design refinement.

Step 1: Determine Over-Arching Goal.

In the first step, the designer determines the goal(s) of the system. The over-arching goal should answer the question, "What is the system trying to achieve?" Any predetermination as to how the task must be accomplished should be excluded. For example, a goal to "obtain milk through a purchase," contains no pre-conceived notion of *how* to purchase the milk. The overall goal should be distilled to only its essential elements–those requirements that are unavoidable without making the scope over-expansive. For example, obtaining milk is a less exclusive goal than purchasing milk. However, broadening the goal beyond solutions under serious consideration is counter-productive (e.g., we would not expand the goal of purchase milk unless we

would consider alternate methods of obtaining milk).

Step 2: Identify High-Level Functions.

The second step is to identify the functions that must be performed to achieve the goal(s). The question to answer at this stage is, "How do we achieve the over-arching goal?" The functions at this stage should be high-level, and, depending on the goal, could consist of only one or a small number of functions. At this point, the designer has not allocated these functions to performing entities. Therefore, the high-level functions must be defined such that they can be allocated to any available entity.

Step 3: Decompose Functions.

Functions are composed of sub-functions in a modular or hierarchical fashion. The high-level functions from step two are decomposed into sub-functions until they reach the atomic function level. *Atomic functions* are functions that can only be performed by a single entity (e.g., it cannot reasonably be decomposed into more than one function where one or more of these functions would realistically be allocated to a human and another would be allocated to a machine). Further decomposition of an atomic function is not desirable, making the determination of automation state a discrete decision. Consequently, all functionality for which the system must account falls under a high-level function. The complexity of a given function depends on the number and interrelationships of its sub-functions.

All non-atomic functions are composed of lower-level functions. There are many proposed methods for decomposing a function, including methods from structured analysis, such as Integrated Computer Aided Manufacturing Definition for Function (IDEF) Modeling [131]. The designer should perform decomposition until functions are indivisible between multiple entities, resulting in *atomic functions*. In practice,

decomposing each function to the point where it is indivisible is not necessary, but instead the designer should decompose each function to the point at which it is impractical to allocate a portion of a function to two separate entities. With system evolution, it may be necessary to readdress the function decomposition as functions which are impractical to allocate to separate entities may change as technology evolves.

The actions taken in step three repeatedly address the question, "Can more than one entity perform function x?" For the purposes of system representation, step three should produce a set of nodes. Although graphical depictions of the atomic functions (such as IDEF diagrams that maintain knowledge of the hierarchical decomposition [132]) are useful, one must take care when naming the functions to insure that all unique functions have unique names.

Steps one through three are similar to the hierarchical task breakdown of methods such as HTA or GOMS but do not include allocation of the detail of user interaction inherent in an allocated function. For example, atomic functions are similar to the leaf nodes created in a hierarchical task decomposition, but functions remain unallocated between human and machine. Therefore, the designer should take special care to ensure an allocation-free breakdown during the first three steps of the present model.

Step 4: Construct Function Relationship Diagram.

After identifying the atomic functions, the functions are transformed into a network by exploring the relationships between the atomic functions. To complete a function, a subset of its atomic functions must be completed in a pre-determined order, as information generated by a function will be an input to other functions. Further, it is common for an atomic function to reside within multiple function hierarchies. These relationships are depicted in a *function relationship diagram* (FRD) wherein the atomic functions represent nodes and the information to pass between

nodes are represented by the connecting arrows. The resulting network provides a temporal ordering similar to that introduced for HAMSTERS [130] for a task network. A function is either mandatory (represented by a solid border in the FRD) or optional (represented by a dashed border in the FRD). Optional functions are those that may need to be performed within some task instances, but not others. Figure 7 shows a legend of the different structures used in the FRD.

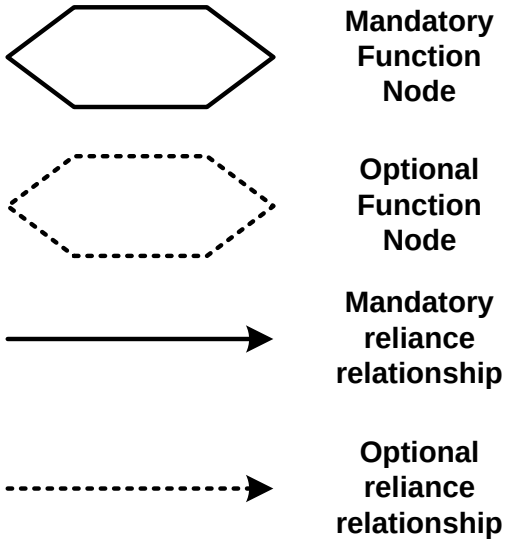

Figure 7. Legend of notation used in Function Relationship Diagram (FRD) structures.

A large number of temporal relationship types can exist between functions. However, each pair of atomic functions can have two possible relationships: dependent or independent. Dependent relationships arise when the completion or product of one task directly influences the other. Dependent relationships are represented by an arrow in the FRD. A function is independent of another when it has no reliance or influence on the completion of the other. All non-connected functions are independent from each other.

Similar to the functions themselves, a relationship is either mandatory (represented by a solid arrow in the FRD) or optional (represented by a dashed arrow in the FRD). A mandatory relationship implies that one function necessarily leads to

the next, while an optional relationship implies a set of circumstances that could avoid this relationship.

Using mandatory and optional functions and relationships, several more-complex relationships, such as those referenced in [124], can be represented. For example, interleaved relationships can be represented as two separate flows within an FRD, with the two functions each with an optional relationship with another common node. An iteration relationship is represented by an optional relationship arrow looping the function node back on itself. The complex relationships are not represented directly, because decomposing these relationships to a lower level allows for further freedom in design.

Multiple relationships may flow into or out of a given node. If an atomic function does not connect to other atomic functions from the higher-level function from which it was derived, the function decomposition should be re-addressed, as this condition violates the rules of the function decomposition. The diagram at this point should not involve the instantiation of function performers (i.e. it is still a *function* relationship diagram and not a *task* relationship diagram).

Step 5: Instantiate functions to tasks.

In step five, the system designer allocates each function to an entity: human or machine. Specific instances of humans and machines are not assigned, we are concerned only *that* a human or machine is performing the function, *not which* human or machine performs it. In fact, to simplify the current discussion, it is assumed that only one human operator is present in the system under design, although this concept could be refined to permit the inclusion of multiple humans interacting with one or more interconnected machines. This step sets a baseline for the states of automation. A *task relation diagram* (TRD) that demonstrates the flow of information from one

entity to another results from this step.

The first step in task instantiation involves induced assignments. Some constraint may mandate the instantiation of a specific function, or set of functions, to a specific entity. Induced assignments can come from rules, capabilities, available resources, or other avenues, but must be addressed no matter the reason for their inclusion. These are assigned first, before any other instantiations are made. Examples of induced functions include decision nodes in systems where humans hold final decision authority or a complex calculation that a human is incapable of performing and a machine must perform.

Once the induced assignments are made, the designer can address the more flexible assignments. The adaptive automation task allocation model discussed in 5 enables the determination of which tasks to assign to a human or machine. By using the model demonstrated in Figure 5, tasks can be assigned to the entity capable of performing the function with maximum proficiency. Although this model may provide insight into which function nodes to instantiate to which entity, it can also draw attention to nodes that are not clearly favored to one entity or the other. The resulting TRD should indicate each node as human or machine, as shown in the legend contained in Figure 8.

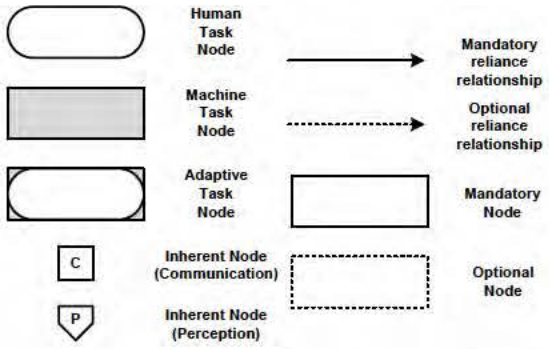

Figure 8. Legend of notation used in Task Relationship Diagram (TRD) structures.

Step 6: Separate Inherent Tasks.

At this step, all functions have been allocated to entities. The resulting TRD consists only of explicit tasks the atomic functions specify. Completion of the task allocation consists of specifying the inherent tasks. Inherent tasks are those tasks that present themselves as the product of a specific task instantiation. However, these inherent tasks can also result from the interactions between the explicit tasks or specific resources available to the system.

The information exchange between entities during a task handoff is the primary source of inherent tasks. Once the designer assigns a task to a machine or human, a set of new complexities emerge through the task relationship diagram: task handoffs. There are four types of task handoffs possible: human to human, human to machine, machine to human, and machine to machine. For the purposes of an adaptive automation system, task handoffs that cross between human and machine are important. A human-to-machine or machine-to-human task handoff requires two inherent tasks that are not present in the underlying functions: *communication* of the information by the losing entity and *perception* of the information by the gaining entity.

Communication of information requires the current performing entity to format the information such that the next entity understands it. For example, a machine that just completed a movie recommendation search task must ensure that it communicates the recommended films to the human before the human can complete the subsequent movie selection task (i.e., displays this information on a screen). On the other end, perception involves the next task performer's ability to obtain and interpret the information communicated to permit subsequent task completion. It's important to make the inherent task nodes in the task relationship diagram visually distinctive. This distinction permits complex task relationships to become more apparent. Step six produces a complete TRD similar to that produced by the previous step, but

including both explicit and inherent tasks. As shown in the legend (Figure 8), the communication and perception tasks of an inherent task are represented by the "C" and "P" nodes that will appear in pairs.

At this point, the designer may find it useful to reiterate through the process to ensure that the diagram truly represents the desired process and system. After this stage, an initial allocation exists. Modeling or prototyping tools can then be used to determine if the human or humans assigned to operate within the system are capable of performing the tasks required from them during typical system operation while having high enough workload to remain engaged with the system. If not, steps five and six are revised until the design attains a desired level of workload. To reduce workload, for example, the human can give a task involved within a complex relationship within the TRD to the machine.

Step 7: Define Adaptive Automation States.

The inherent tasks of communication and perception provide one of the most important steps in designing an automation system. When the designer adds adaptive automation to the system, an understanding of cognitive task handoffs is crucial. The selection of a set of atomic tasks, or groups of tasks, in the TRD to become adaptive nodes makes the automation adaptive, and consists of identifying nodes that can switch between human and machine instantiation based on some pre-defined trigger.

Selecting a given node as an adaptive node, adds complexity to the overall system. As a node switches from static human or machine to adaptive, the number of handoff types needed can double for each outgoing connection. For example, a human node connected to a machine node requires one type of handoff, a human node connected to an adaptive node requires two, and an adaptive node connected to an adaptive node requires four. It should be noted, however, that this added complexity only

occurs for nodes that are independently switched within the design. It is possible for a group of nodes to always be switched in concert with one another as a cluster, in which case, only the number of connections between this group of nodes and the nodes they connect with are doubled.

Step seven finalizes the selection of adaptive nodes. As shown in the legend (Figure 8), adaptive nodes are represented by a shape that combines those of the human and machine nodes. Choosing adaptive nodes is ultimately a subjective task. However, an analysis of the TRD can help make these decisions easier and more grounded. Five analysis tools include: determining the number of possible states, node clustering, task handoff analysis, branch counting, and inherent task load comparison. By iterating through these tools, an adaptive automation system emerges.

Number of possible states.

Once the designer selects adaptive nodes, they must readdress the complexity and handoffs created through the selection. One way to assess complexity involves determining the number of possible automation states. For each independently adaptive automation node in the relationship graph, two possible states exist: human and machine. Therefore, the number of possible states equals 2^x, where x is the number of adaptive nodes in the current design. Fewer independently adaptive nodes means exponentially fewer possible states. Within the TRD, one can simply count the number of desired adaptive nodes and plug it into the above equation. This is a rough approximation of the potential challenges.

Task handoffs.

Related to the total number of possible states, one way the TRD can help analyze the designed system is through an analysis of the task handoffs–specifically the num-

ber of different-entity handoffs. In each possible case where a machine hands off to a human or human to a machine, count one handoff. In the case where a node is set as adaptive, this implies that a handoff from an adaptive node to another adaptive node counts twice, while an adaptive to non-adaptive node handoff will count once. This handoff count suggests the number of nodes where task load shifts from one entity to another. By focusing on these nodes, potential bottlenecks appear due to certain handoff tasks taking place more often than at other locations. Highlighting these tradeoffs allows the system designer to visualize the locations where inherent tasks–specifically those associated with communication and perception, as described in Section 2.3–reside.

Node clustering.

Functions clustered based upon the degree of the edges in and out of a given node tend to provide similar or highly inter-related functions. Therefore, automation of a cluster as a set can often be achieved with greater effect than just automating one function in the set. Conversely, the designer could also instantiate all of the tasks in a cluster to a human, since the information associated with the multiple edges will not need to be exchanged.

An example of this would be in piloting an aircraft. Although the "takeoff function" contains many lower level functions, there are many complex groups of functions within it that naturally group together to ensure a proper amount of situation awareness, where the necessary information is provided through the right kinds of feedback.

Within the TRD, an adaptive node cluster can be represented by a large adaptive node surrounding a set of tasks. This implies that all of the tasks within the adaptive node cluster will all always have the same allocation. This allows for adaptation without increasing the number of states exponentially. Additionally, since a hierarchical

notation was not kept in Steps 1-4, the TRD now has flexibility to cluster in unique ways not easily perceivable through the hierarchical breakdown.

Branch counting.

Branch counting refers to the idea of determining the number of other atomic tasks to which one specific atomic task connects. A task that influences or is influenced by a large number of tasks makes automation more difficult. This is because changing a node to adaptive requires an inherent communication/perception node to each incoming and outgoing relationship. For example, forming a node with one outgoing relationship and one incoming relationship adaptive creates two new communication/perception nodes, while doing the same to a node with one incoming relationship and four outgoing relationships creates five new communication perception nodes.

Tasks that have large branch counts can often make good candidates to be members of an adaptive node cluster. Conversely, single branches within a TRD can often indicate good locations to place adaptive nodes, as the inherent task load is likely smaller. Although some individual task handoffs may be very difficult, branch counting a TRD provides a good snapshot of where there will be a large number of task handoffs that the designer may not have foreseen.

Inherent task load comparison.

An inherent task load comparison provides another means to analyze the effectiveness of design options. This consists of a comparison of the relationship diagrams created when the TRD instantiates one function as a human task versus when the TRD instantiates the same function as a machine task. The primary difference as a result of this reallocation is the number of inherent tasks that are added or subtracted.

A comparison of the two instantiations helps to visually communicate inherently understandable design decisions.

2.4 Function to Task Process Illustrated

The function-to-task process model is illustrated here by designing an adaptive automation system to aid a user during play of Space Navigator. The result of the process is an allocation of tasks to aid adaptive automation system design. The dynamism of elements within Space Navigator (e.g. movement of no-fly zones, random appearance of new ships, etc.) do not allow for creating an "optimal" automated player. Because the game is relatively simple in its mechanism while still difficult for a machine to perform optimally, Space Navigator is a good environment for illustrating the process model. Each of the seven steps of the Function-to-Task Design Process Model (Figure 6) is addressed in the following sections.

Step 1: Determine Over-Arching Goal.

To fulfill Step 1, we ask the question "What are we trying to achieve?" The goal of a *Space Navigator* player is to *score the most possible points*. In the present case this is a simple task, since the goal is clear from the rules of the game. However, in other situations a goal may be more difficult to define. For this reason, the loop from Step 2 to Step 1 may provide further insight and refinement for more complex systems.

Step 2: Identify High-Level Functions.

With the goal in hand, we ask "How do we score the most possible points?" This helps identify the high-level functions as the manners in which points change. Four high-level functions become apparent:

1. Move spaceship to intended target planet.

2. Pick up bonuses.

3. Avoid collisions with other spaceships.

4. Avoid traversing no-fly zones.

Step 3: Decompose Functions.

Answering the question, "Can we further divide function x?" allows further function decomposition. After applying this decomposition we obtain the following list of atomic functions:

1. Function 1: Move spaceship to intended target planet.

    - Determine the best ship to draw route.
    - Identify destination planet of selected ship.
    - Identify if ships have routes already.
    - Create a set of possible routes.
    - Select a route.
    - Draw a line from selected ship to destination.

2. Function 2: Pick up bonuses.

    - Identify all available, non-selected bonuses.
    - Identify destination planet of selected ship.
    - Determine if route change to pick up bonus is worth points gained.
    - Determine if selected ship has a route already.
    - Adjust route to pick up bonus.

3. Function 3: Avoid collisions with other spaceships.

    - Detect likely collisions.
    - Identify destination planet of selected ship.
    - Determine if selected ship has a route already.
    - Determine if route change to avoid collision is worth points gained
    - Adjust route to avoid collisions.

4. Function 4: Avoid traversing no-fly zones.

    - Identify no-fly zones.
    - Identify ships headed toward a no-fly zone.
    - Identify destination planet of selected ship.
    - Determine if selected ship has a route already.
    - Determine if no-fly zone traversal is worth lost points.
    - Adjust route around no-fly zone.

This atomic function list demonstrates two concepts previously discussed in Section 2.3: the overlap of specific atomic functions and the circumstance-specific nature of atomic functions. Some atomic functions in the above list appear in multiple locations within the hierarchy. For example, the atomic function "identify destination planet of selected ship" appears in all high-level functions. It is the same function and named the same in every case. Secondly, a practitioner may consider the atomic functions listed above as more complex depending on the interpretation of the process. For example, under the avoid no-fly zones high-level function, the atomic function "Determine if no-fly zone traversal is worth lost points" could be considered a non-atomic function made up of sub-functions such as "determine the number of potential

points lost," "determine amount of time added," "determine increased collision likelihood," etc. However, the designer must ask whether they would consider dividing these tasks among human and machine entities or whether they would always assign them to the same entity. An important note from Step 3 is that moving further along the Process Model may provide insight into the proper level to end Step 3. This is represented explicitly by a revision loop from Step 4 back to Step 3 in Figure 6.

Step 4: Construct Function Relationship Diagram.

We now produce the functional relationship diagram by analyzing each unique atomic function in relation to all of the other functions and assigning relationships (dependent or independent) based upon the transfer of information from one function to another. The end result of this relationship mapping is the function relationship diagram shown in Figure 9. The creation of this diagram illustrates overlapping atomic function instances, optional versus mandatory functions, and the flexibility provided for structuring the functions by removing the hierarchical structure.

In this step, there are a few instances where multiple higher-level functions contain the same atomic function (e.g. "identify destination planet of selected ship"). Only one node in the functional relationship diagram represents these functions. However, these functions interact with many different functions. The "identify destination planet of selected ship" function directly influences three separate functions. Therefore, functions that overlap multiple higher-level functions provide potential bottlenecks in the relationship diagram. That is, the information these functions produce must be available to any human or machine entity to permit subsequent functions' performance.

Optional and mandatory functions and relationships are all represented. For example, we need to identify potential collisions (mandatory function) and must do it

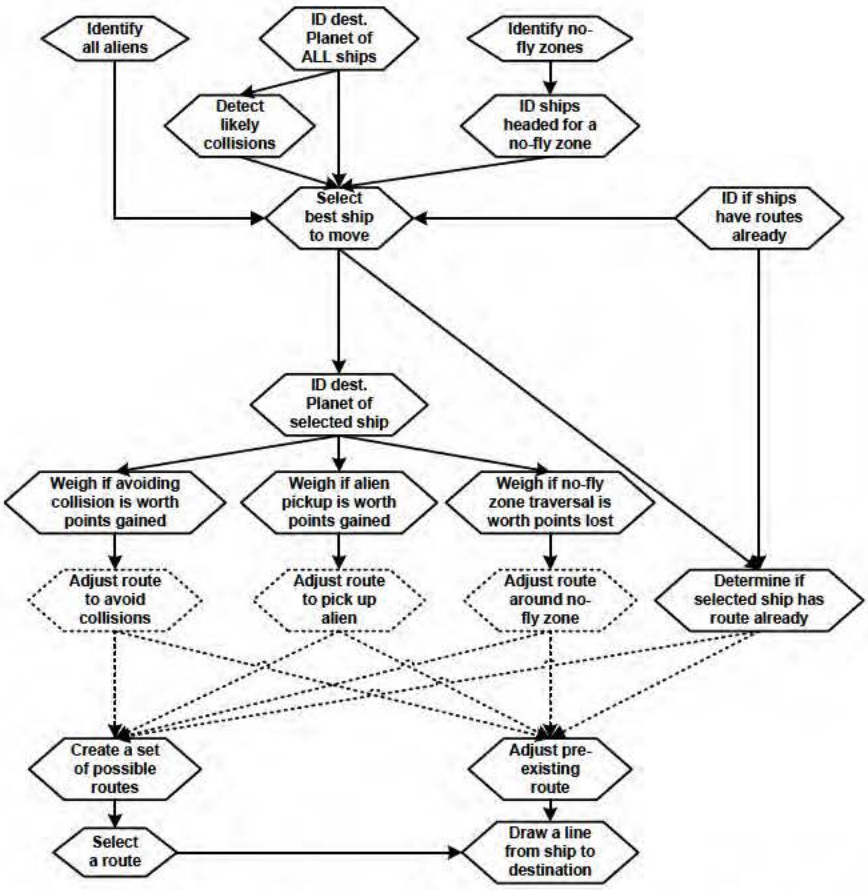

Figure 9. Functional relationship diagram of the Space Navigator game. See Figure 8 for a legend of notations used.

before adjusting our route for them (mandatory relationship), but this information may not necessitate a route adjustment (optional function). Then the ensuing adjustment is made in one of two ways depending on whether the selected spaceship already has a route (optional relationship).

Relationships that seem compartmentalized in the function decomposition can appear highly interconnected in the relationship diagram. The four high-level functions identified for the Space Navigator game are separated distinctly in the functional decomposition in Step 3, within the hierarchical structure. However, when the hierarchical structures are removed, the sub-functions provide a system that cannot be easily divided along the lines of the previously defined high-level functions. This

change in perspective can also influence a different understanding of the high-level functions themselves. For example, looking at the resulting FRD a designer could potentially conclude that the tasks of "spaceship selection" (the upper half of the diagram), "action decision" (the bottom half of the diagram), and "action performance" (the final function) are the higher-level functions. This demonstrates how the FRD provides a revision path that can lead the practitioner back to Steps 2 or 3 for further analysis.

Step 5: Instantiate functions to tasks.

The design goal in this example is to apply automation to aid the user when interacting with this game where the assumed default state is that the human operator will perform all functions. Therefore, the goal of the function allocation in this particular example is to identify alternate automation states, which will permit an operator to perform well in the presence of exceptionally high spaceship spawn rates. Referring to Figure 9, one can see that the functions in the center of the diagram are highly inter-connected. This interconnection implies that the human and machine would need to exchange significant amounts of information if elements within this region of the figure were divided between these entities. However, other elements near the periphery of the diagram are not as highly interconnected. As a result, allocation of many of these elements to the machine are likely to result in less need for communication between the human and machine.

Based on this analysis and the performance of the human and machine, Figure 10 represents a potential task allocation and resulting task relationship diagram. In the diagram, tasks that the machine controls are those that appear as gray boxes and those that the human controls are represented by rounded boxes. Note that Figure 10 shows a TRD after Step 6 is complete.

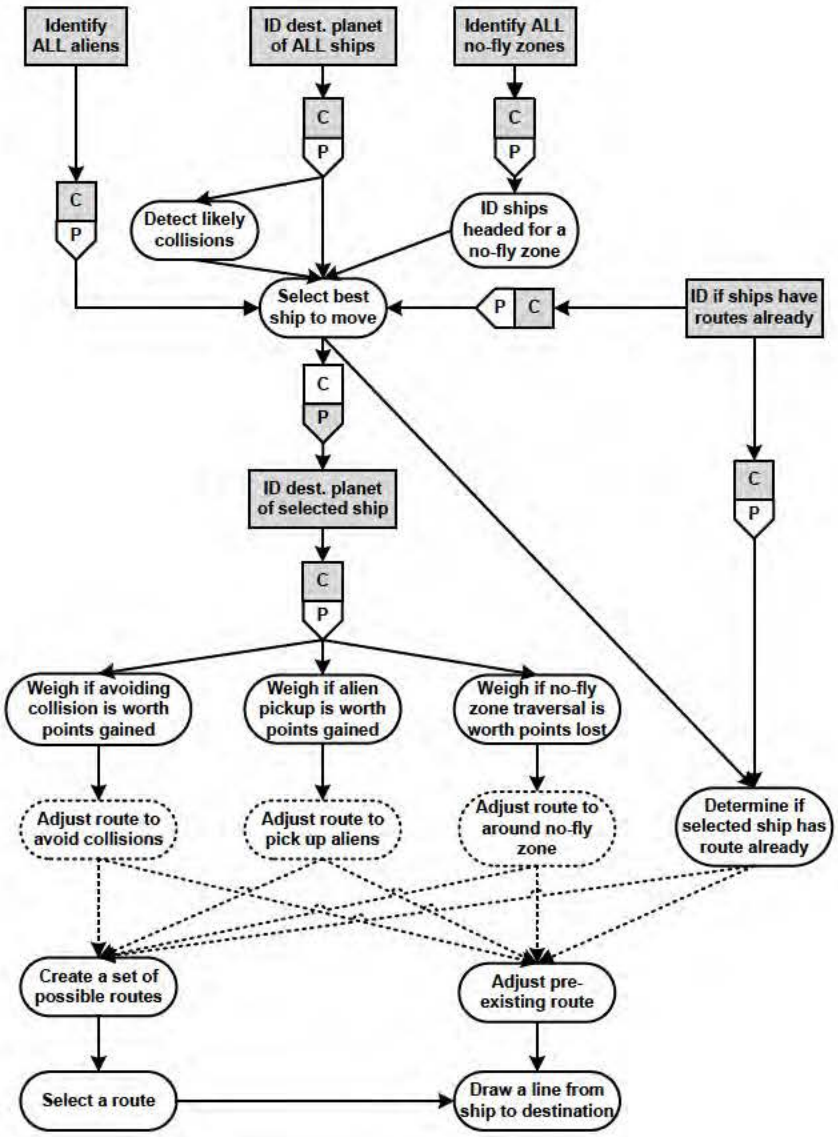

Figure 10. Task relationship diagram of the Space Navigator game with inherent tasks shown, where functions have now been allocated to human and machine. See Figure 8 for a legend of notations used.

Step 6: Separate Inherent Tasks.

As discussed earlier, the "C" and "P" nodes are also shown in Figure 10. These nodes indicate the need for the entity performing the function near the "C" node to perform the inherent task of communicating to the receiving entity and the need for the entity performing the function near the "P" node to perform the inherent

task of perceiving and interpreting the information to enable performance of the function. Because of the selected function allocation, there are several task handoffs from human to machine and vice versa as the "C" and "P" nodes indicate. Some of the communication/perception chains are inconsequential, like communicating from the machine to human if a specific spaceship has a route already–as a simple path is already drawn between entities to aid transfer of this information. However, others are more difficult.

For example, communicating the destination planet for all ships to a human can be simple, but it is important and perhaps difficult to ensure perception. If the ships are color-coded to align with a specific planet, this task is simple for most people when few entities are available. However, it becomes increasingly difficult as the number of entities increase and in some cases impossible for certain individuals (e.g., those who are color blind). Therefore, it is not only needed to communicate the information, but to confirm the transfer of critical information to insure a handoff. Steps 5 and 6 are interconnected. The Process Model (Figure 6) has a revision loop connecting the latter to the former, as the inherent tasks that appear in Step 6 can inform decisions in Step 5.

Step 7: Define Adaptive Automation States.

Finally, we apply adaptive automation to the TRD. Figure 11 shows how the process changes after adaptive automation nodes are selected. Although all of the tools presented in Step 7 are useful, some will prove more useful in specific situations than others. For *Space Navigator*, the number of possible states and task handoffs are implied by application of the other tools: node clustering, branch counting, and inherent task comparison.

Looking at the original TRD in Figure 10, a few groupings of tasks begin to appear.

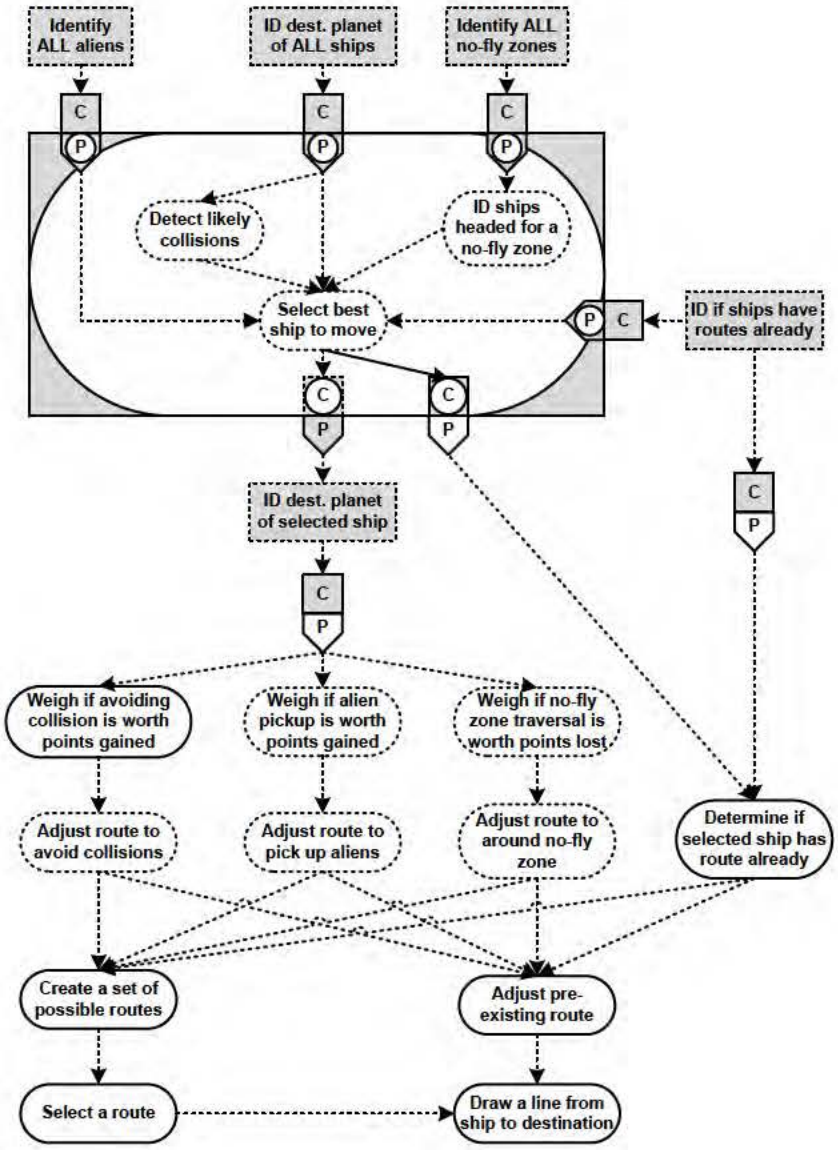

Figure 11. Task relationship diagram of the Space Navigator game, where functions have now been allocated to human nodes, machine nodes, and adaptive nodes. See Figure 8 for a legend of notations used.

One shows up near the top that encapsulates several spaceship selection related tasks, while a second group near the bottom represents several action-oriented tasks. These groups of tasks could prove useful as adaptive node clusters.

Branch counting is then applied to narrow the results from our node clustering. The action-oriented cluster is eliminated due to the large number of branches present

in this location of the diagram. From a more subjective perspective, it would obviously be difficult to automate these tasks as they tend to rely on dynamic elements of the game (e.g. no-fly zones and ships that move).

The final TRD in Figure 11 arises from inherent task load comparison. The spaceship selection cluster is further refined by analyzing where the inherent tasks cross. By looking at these tasks, we see that there is a group of tasks that are all controlled by the same entity. Thus allowing a set of similar communication and perception inherent tasks. The large cluster then is chosen for an adaptive automation.

In this case, one can imagine a system that adapts to the player's workload to automate the selection of which ship to act on when the user is inundated with choices, by highlighting a specific ship. This arrangement permits the system to decide which spaceship is most important to route or reroute and conveys this information, ideally in a very clear fashion, to the human. The human selects and draws the path. As such, the system determines the critical ships to address, a task that can be difficult for the human when the screen is cluttered, while allowing the human to select a route, a task that is too complex for the machine to perform reliably. It is important to note the revision loops in the process model once again here. Once complete, the design may require a complete overhaul or several minor tweaks.

2.5 Conclusions

A function-to-task design process model has been presented that assists an adaptive automation system designer in determining the allocation of tasks between the human, machine, and dynamically between the two. This process model permits the designer to investigate the effects of allocation on explicit and inherent task load for the intended user when interacting with an adaptive automation system. The process model demonstrates that reallocation of functions imposes a change in inherent tasks

to permit the proper communication and perception of information between the performing entities. As such, reallocation of a function implies a change in information flow between the machine and human, and this change requires the allocated task performer to utilize cognitive and physical resources to communicate and perceive the appropriate information to enable task performance.

Consideration of the information available in the task relationship diagrams when performing task allocation permits the designer to understand and potentially reduce the volume or complexity of information exchange between a human and machine. This tool may also help to reduce unwanted redundancy between the functions the human and the machine perform by clarifying the form of the information necessary to facilitate human decision making.

The function-to-task model requires the designer to identify the functions that are necessary to achieve the goals of the system and decompose these functions into leaf-level or atomic functions. The dependencies among these functions are then explicitly captured in a function relationship diagram. The functions are then allocated to an appropriate entity to form the basis of a task relationship diagram. Information flow between independent entities is then defined, identifying inherent tasks that are present in the allocation to provide communication. The form of the task relationship diagram is then evolved through application of five analysis tools to identify points in the TRD where adaptive automation could be easily accommodated. The application of this process model thus results in the allocation of tasks to the human, machine, or dynamically to the two entities.

The TRD resulting from a systematic implementation of the function-to-task design process model allows the designer to identify locations within a system where adaptive automation could provide benefit. However, this process model does not result in the design of an adaptive automation system but aids the designer in during

the conceptual design portion of the user-centered design process [110]. Coupling the information from the task relationship diagram with the existing adaptive automation taxonomy, the designer can more effectively create well-targeted adaptive automation systems. Further, the tasks derived from this process can be used as input to existing interface design models, such as DIANE+ [122] or ConcurTaskTrees (CTT) [124], which enable detailed design of the user interface.

III. Clustering-Based Real-Time Player Modeling

To move from the Function to Task Design Process Model to an adaptive automation implementation, the automation must be built; an automation must first exist, before it can become an adaptive automation. For this research, an adaptive automation node needed to be created within the *Space Navigator* environment's task relationship diagram. The trajectory draw task was chosen as a task that could be performed either by the machine or by the human, thus lending itself to adaptive automation.

This chapter presents work that describes the trajectory drawing automation system. The work presented here began with an off-line player modeling system to create trajectories, which was presented at the *28th Canadian Conference on Artificial Intelligence*,[1] However, this work was later adjusted to work as a real-time trajectory generator. The following is a slight adaptation to a paper submitted to the *IEEE Transactions on Computational Intelligence and AI in Games*,[2] with work presented in previous chapters of this dissertation omitted.

3.1 Introduction

Automating game-play in a human-like manner is the goal of a large area of intelligent gaming research, with applications from trying to succeed in a gaming version of the "Turing Test" [133] to creating human-like game avatars [134]. When we move from playing a game like a generic human to performing like a specific human, the dynamics of the problem change [135]. Generalized datasets can no longer be lumped into large groups of past game-play. In complex dynamic environments it can be

[1]Bindewald, J. M., Peterson, G. L., and Miller, M. E. Trajectory generation with player modeling. In *Advances in Artificial Intelligence* (2015), Springer, pp. 42–49.

[2]Bindewald, J. M., Peterson, G. L., and Miller, M. E. Clustering-based real-time player modeling. *IEEE Transactions on Computational Intelligence and AI in Games* (SUBMITTED).

difficult to differentiate individual players, because the insights exploited in imitating "human-like" game-play can become less useful in imitating the idiosyncrasies that differentiate specific individuals' game-play.

There are several benefits of individual player imitation. Individual player imitation provides insights into modeling more believable opponents [134]. Better understanding what sets individual players apart from others, allows a game designer to build more robust game personalization [136]. Learning the aspects of a game state that set individual players apart, allows for better understanding of how to adjust games according to skill level [137].

This paper contributes a real-time individual player modeling system that enables an automated agent to perform response actions in a game that are similar to those that an individual player would have performed in similar situations. This work improves upon past player modeling efforts, such as [138], emphasizing three things. First, the player modeling system automatically updates in real-time rather than requiring off-line computation to adjust to changing game-play over time. Secondly, the system takes advantage of insights gleaned from past game-play clustering to gain a statistically significant differentiation between players in a relatively short amount of game-play (five, five-minute games). Additionally, the clustering-based player modeling method allows the practitioner to glean insights into what differentiates the game-play of individual players.

This paper proceeds as follows. Section 3.2 reviews related work in the fields of player modeling and learning from past game-play. Section 3.3 presents a generic player model methodology and then uses the generic player model as a base for implementing a real-time individual player modeling system. This model is then demonstrated using the *Space Navigator* trajectory generation game as a test-bed. Section 3.5 gives experimental results showing the individual player modeling system's

improvements over the generic modeling method. Section 3.6 summarizes the findings presented and proposes potential future work.

3.2 Related Work

Player modeling research informs the methodology to create trajectories similar to those of an individual player. Methods involving learning from past experiences provide insight into how to generate new trajectories from past game-play instances. This section describes some over-arching areas of past work that influence the current research.

Player Modeling.

Three taxonomies for player modeling exist, each providing a different way of organizing the field. Each model is presented and explained. Interspersed with the model descriptions are examples of how the model would classify different player modeling research efforts.

Yannakakis Model.

In the Yannakakis player model taxonomy [139], four input types are used to build player models of two types which provide four types of outputs. Inputs to a player model fall into four categories: game-play, objective, game context, and player profile. Game-play data (often called behavioral data) captures actions that a player takes in the given game environment. Objective data includes a player's measurable physiological responses to the game environment. Game context data denotes a representation of the real-time state of the game. A player profile is a static representation of the player outside of the context of the game (e.g. personality type). These four inputs are used in some combination to create a player model.

The resulting player model is either a model-based (top down) or model-free (bottom-up) player model. In a model-based player model the model is built on some form of a theoretical framework where player groupings are pre-defined according to some set of features. Examples of model-based player models include supervised neural networks [140], trait theory to pre-determine player types [141], strategy groupings based on game design features [142], and association rule mining to find player experience/activity relationships [143]. In model-free player models the goal is to find player types that naturally arise from the collected data. Clustering is a common method to find player types, some examples of which include hierarchical clustering [144], k-means [145, 146], neural networks [140], and self-organizing maps [144].

When utilized, player models produce an output when a given state or response is presented to it depending on its intended purpose. The outputs from player models can encompass scalar values, class membership, ordinal data (rankings), or no output (such as when learning a player model for clustering purposes).

Smith Model.

The Smith player model taxonomy [147] classifies player models across four independent facets: domain, purpose, scope, and source. The domain of a player model is either game actions (similar to Yannakakis's game-play data input type) or human reactions (similar to the objective and player profile input types). The second facet, purpose, describes the end for which the player model is implemented: generative player models aim to generate actual data in the environment in place of a human or computer player, while descriptive player models aim to convey information about a player to a human. The scope of the player model describes the scope of players the model represents: individual (one), class (a group of more than one), universal (all), and hypothetical (some theoretical player or set of players that doesn't fit in the other

categories). The source of a player model can be one of four categories: induced - objective measures of actions in a game; interpreted - subjective mappings of actions to a pre-defined category; analytic - theoretical mappings based on the game's design; and synthetic - based on some non-measurable influence outside of the game context (e.g. hunches).

For descriptive purposes, each player model is given a type for each of the four facets. For example, the player model created in [148] for race track generation models individual player tendencies and preferences (Individual), objectively measures actions in the game (Induced), creates tracks in the actual environment (Generative), and arises from game-play data (Game Action).

Bakkes Model.

Bakkes *et al* [149] create a player behavior model that classifies player models that involve game-play data inputs in the Yannakakis model or fall in the game action domain in the Smith model into four categories:

- Player behavior models based on *player actions* map states encountered in the game to player actions. A good example of this type of model is the player models associated with research on poker player modeling [150].

- Player behavior models based on *player tactics* take multiple actions and/or the actions of multiple players into account to model different players, an example being the tactical offensive football play models created in [151].

- Player behavior models based on *player strategies* involve the use of different tactics in succession, and tend to account for "entire game" time frames. Examples of strategy level player behavior modeling exist in real-time strategy game research, such as systems designed to play *StarCraft* [152, 153].

- *Player profiling* involves the use of player behavior in games to establish psychological or sociological player profiles. Research efforts measuring entertainment in games [154, 155] tend to allow for this type of behavior modeling.

Learning from Previous Game-Play.

Two areas of research that rely on past experience to inform future automated game-play include: Case-Based Reasoning (CBR) and Learning from Demonstration (LfD). Both CBR and LfD train an automated system to generate responses based on observations within an environment. In CBR, a "case-base" maintains a set of observed environment states and their associated responses (cases) [156]. When a new state is received by the CBR a previous case is retrieved, adapted to the current state, and a new response is fashioned. The new state and its associated response is then either added to the case-base or thrown away according to observed feedback. In LfD, a teacher demonstrates a skill that it would like the the automated system to learn [157]. The learner attempts to derive a policy based on the demonstration, and then attempts to execute the derived policy. The policy is then evaluated and updated with feedback in the environment.

The nearest neighbor principle maintains that instances of a problem that are a shorter distance apart more closely resemble each other than do instances that are a further distance apart [158]. This concept is applied in many locally weighted learning algorithms that learn how to perform regression or classification tasks by comparing an incoming instance to that of its nearest neighbors [159]. The nearest neighbor principle is used to find relevant past experiences in LfD tasks such as a robot intercepting a ball [160], CBR tasks such as a RoboCup soccer-playing agent [161], or tasks integrating both LfD and CBR such as in real time strategy games [162]. When searching through large databases of past experiences approximate nearest neighbors

searches, such as Fast Library for Approximate Nearest Neighbors (FLANN [163]), have proven useful in approximating nearest neighbor searches while maintaining lower order computation times in large search spaces.

3.3 Methodology

The real-time player modeling paradigm we present here improves on previous work in three ways. As the name suggests, the player model updates in real-time to adapt to changing player habits. Additionally, the paradigm pulls insight by clustering past game-play, differentiating between players quickly. Then, the resulting player models allow the practitioner to investigate specific individual game-play tendencies further. Figure 12 illustrates our real-time player modeling paradigm. This real-time player modeler creates responses to provided game states that are similar to those that an individual player would have given in response to similar states. This section explains the three main tasks of the real-time player modeler: creating a generic player model, generating similar response trajectories, and real-time updating of the player model with individual player game-play.

Create Generic Player Model.

This section outlines the generic player model creation process, shown in shaded area 1 of Figure 12. Clustering the past game-play instances both by state and response reveals general player tendencies. With information gained from clustering, pruning outlier instances creates a universally representative example game-play dataset. This dataset then forms the groundwork for a generic player model that maps state clusters to response clusters.

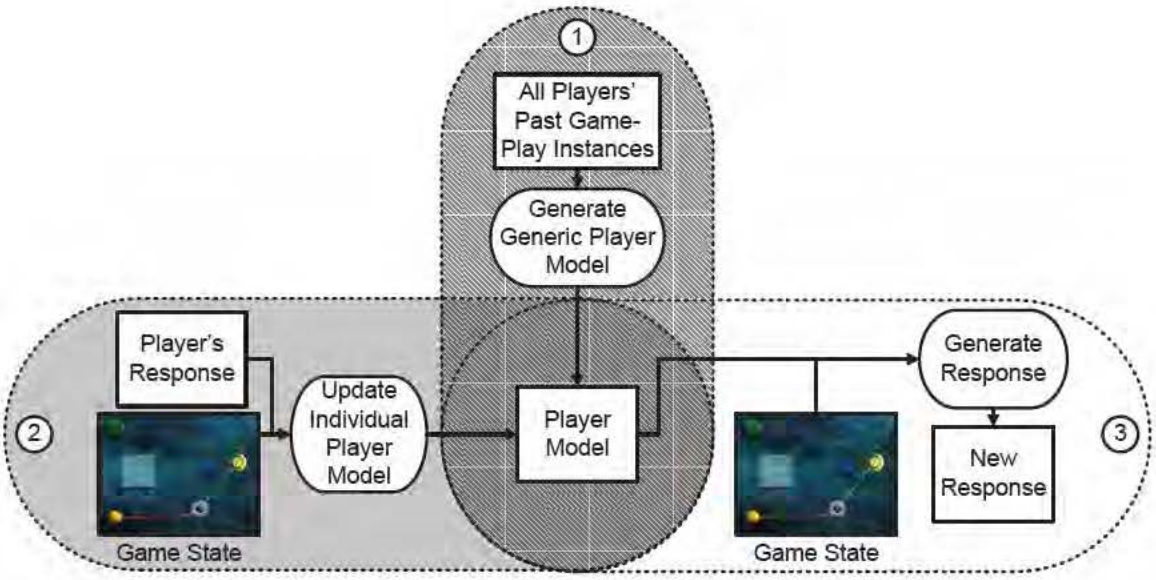

Figure 12. A real-time updating individual player modeling paradigm.

State and Response Clustering.

Similar to the method used in [138], Ward agglomerative clustering [164] provides a baseline for player modeling. Clustering reduces the state-response pairs into a set of representative clusters, reducing the potential representation size of a player model. This method was proven effective for clustering in a trajectory creation game environment in [138, 165]. Agglomerative clustering starts with a set of game-play instances that contain a state and its associated response and assigns each instance to a state cluster and a response cluster. The number of clusters will depend on the environment and size of the underlying dataset. The mapping from a state cluster to a response cluster for state-response instance demonstrates a proclivity for a player to react with a given maneuver in a specific type of game situation. By determining the frequency of state cluster to response cluster mappings, common situational responses and outlier actions emerge.

Cluster Outlier Pruning.

The frequency of state cluster to response cluster mappings reveal common and outlier situational responses, providing the basis for two types of pruning that generalize the generic player profile by removing infrequent interactions. First, instance frequency within clusters helps in pruning outliers from the set of all game instances. If a given state has only been seen in one instance by one player, that state is unlikely to provide much benefit in predicting future responses. Similarly, a response given by only one player in one instance is unlikely to be replicated in future player responses.

Clusters with outlier responses are removed first by removing all instances assigned to the least populated response clusters. The cutoff threshold for determining which instances to remove could be either a minimum response cluster size or just a percentage of response clusters to remove. For example, due to the distribution of cluster sizes in the *Space Navigator* database we removed instances falling in the bottom 25% of all response clusters according to cluster size. Setting cutoff thresholds relies on knowledge of the environment and underlying dataset distribution.

Next, outlier state clusters are removed in two ways. First, instances that fall in the bottom 25% of all state clusters according to cluster size are removed, removing all clusters that are rare in general. However, removing states not seen by many *different* players is also important. In addition to removal based on sheer cluster size, pruning also removes instances falling into a state cluster encountered by a minimal subset of players. This removes a subset of clusters not removed previously: state clusters with many instances reached by an extremely small subset of players. It is important to note that although the generic player model will be built on the pruned game-play database, the original state and response clusters are used. In this way, individual player modeling can still capture outlier states or responses individual players encounter that most players do not encounter.

Player Model Creation Algorithm.

Whereas [138] used an offline game-play database creation method, the current research uses a faster online system to model players. Algorithm 1 generates a generic player model that determines the likelihood each state cluster is to map to each response cluster.

Algorithm 1 Generic player model creation algorithm.

1: **inputs:**
2: x = the number of state clusters
3: y = the number of response clusters
4: $\mathbf{M} = \{\langle S_1, R_1\rangle, \langle S_1, R_2\rangle, \cdots, \langle S_x, R_y\rangle\}$, all state-response cluster mappings

5: $\mathbf{C} = \boldsymbol{O}_{x,y}$ ▷ $x \times y$ zero matrix
6: $\mathbf{P} = \boldsymbol{O}_{x,y}$ ▷ $x \times y$ zero matrix
7: **for** $i = 1 \rightarrow x$ **do**
8: **for** $j = 1 \rightarrow y$ **do**
9: $c_{i,j}$ = the number of instances assigned to cluster mapping $\langle S_i, R_j\rangle$
10: **end for**
11: **end for**
12: **for** $i = 1 \rightarrow x$ **do**
13: **for** $j = 1 \rightarrow y$ **do**
14: $\mathbf{P}_{i,j} = c_{i,j} / \sum_{k=1}^{y} c_{i,k}$
15: **end for**
16: **end for**
17: **return P**

The generic player model creation algorithm takes in the number of state and response clusters (x and y respectively), and the set of all state-response cluster mappings ($\mathbf{M}$). Line 5 creates a matrix of counters ($\mathbf{C}$) to help determine how many instances belong to each cluster in $\mathbf{M}$. Line 5 creates an empty player model ($\mathbf{P}$) that will hold likelihoods for each state-response cluster mapping in $\mathbf{M}$. Both $\mathbf{C}$ and $\mathbf{P}$ are initialized to the $x \times y$ zero matrix. The loops beginning in Lines 7 and 8 process each of the state-response cluster mappings. For each cluster mapping, the number of instances provided by players that belong to cluster pairing $\langle S_i, R_j\rangle$ is recorded.

The loop beginning in Line 12 creates the player model **P** from the counter matrix **C**. The model contains a matrix of likelihoods that a given instance provided by a generic player chosen at random from the game-play database will belong to the indicated state and response cluster mappings. The likelihoods are determined by normalizing across the rows of **C**. For each state cluster, the count for each response cluster is divided by the total number of instances assigned to the state cluster. The matrix of likelihoods is returned as the generic player model **P**. This generic player model forms the baseline for individual player model creation. To model individual player gameplay habits, the individual player modeling techniques in the next section update the generic player model through observed game-play data.

Update Individual Player Model.

For real-time individual player modeling, this research updates the generic player model created in Algorithm 1 as an individual plays the game. Over time, the updates shape a player model that represents an individual player's game-play tendencies, as illustrated in shaded area 2 of Figure 12. The individual player update process involves an algorithm to learn a player model with individual player tendencies over time. In order to train an individual player model quickly, the information gained from each state-response instance leads to an update of the state-response cluster scores.

Individual Player Model Real-Time Update Algorithm.

Algorithm 2 demonstrates the real-time updates that take place to learn an individual player's tendencies. The algorithm begins with the generic player model **P**. Once a player submits a response in the game environment, the current game state and the response are submitted. The algorithm finds the closest state (S_{close}) and re-

sponse (R_{close}) clusters to the state and response passed in by the player. The player model is updated at the intersection of S_{close} and R_{close} by δ_{close}. Then the player model is normalized across all the R values for S_{close} so that the values sum to 1.

Algorithm 2 Individual player model real-time update algorithm.

1: **inputs:**
2: $\mathbf{P}$ = an $x \times y$ generic player model created by Algorithm 1
3: $\langle s_{in}, r_{in} \rangle$ = a player-provided state-response pair
4: $\mathbf{M} = \{\langle S_1, R_1 \rangle, \langle S_1, R_2 \rangle, \cdots, \langle S_x, R_y \rangle\}$, all state-response cluster mappings

5: S_{close} = the closest state cluster to state s_{in}
6: $\delta_{close} = q \cdot (\delta_{cp} + \delta_{cmv} + \delta_{pma})$, S_{close}'s update increment weight
7: R_{close} = the closest response cluster to response r_{in}
8: $\mathbf{P}(S_{close}, R_{close}) = \mathbf{P}(S_{close}, R_{close}) + \delta_{close}$
9: **for** $\mathbf{P}(S_{close}, i)$ where $i = 1 \rightarrow y$ **do**
10: $\quad \mathbf{P}(S_{close}, i) = \mathbf{P}(S_{close}, i) \,/\, (1 + \delta_{close})$
11: **end for**

Update Increment Weighting.

The player model update algorithm is useful in modeling player behavior, but there are certain states from which more can be gleaned than others. Weighting the increment values for a given state-trajectory pair can be useful in quickly learning idiosyncrasies that set a player apart from the generic player. Specifically, knowing which state clusters contain the most information for future player modeling is useful. Traits gleaned from the clustered data provide ways to help determine which state clusters should create larger learning increments, and which states provide minimal information to extend beyond the generic player game-play model. Three binary traits contribute to the update increment, δ, in Line 6 of Algorithm 2. The three traits calculated to help weight δ include cluster population, cluster mapping variance, and previous modeling utility.

Cluster Population: When attempting to learn game-play habits quickly, knowing the expected responses of a player to common game states is important. Weighting

δ according to the size of a state cluster in comparison to that of the other state clusters across the entire game-play dataset emphasizes increased learning from common states for an individual player model. States that fall into larger clusters can provide better information for quickly learning how to differentiate individual player game-play habits. To calculate the cluster population trait, all state cluster sizes are calculated and a population threshold is selected. Any state cluster with a population above the population threshold is given a cluster population trait weight of $\delta_{cp} = 1$ and all other state clusters receive a weight of $\delta_{cp} = 0$.

Cluster Mapping Variance: When mapping state clusters to response clusters, some state clusters will consistently map to a specific response cluster across all players. Other state clusters will consistently map to several response clusters across all players. Very little about a player's game-play tendencies is learned from these two types of state clusters. However, state clusters that map to relatively few clusters per player (intra-player cluster variance), while still varying largely across all players (inter-player cluster variance) can help quickly differentiate players. The state cluster mapping variance ratio is the total number of response clusters to which a state cluster maps across all players divided by the number of response clusters to which the average player maps, essentially the ratio of inter-player cluster variance to the intra-player cluster variance. The cluster mapping variance trait weight, δ_{cmv}, is set according to a cluster variance ratio threshold. All state clusters with a variance ratio above the threshold receive a weight of $\delta_{cmv} = 1$ and all others receive a weight of $\delta_{cmv} = 0$

Previous Modeling Utility: The last trait involves running Algorithm 2 on the existing game-play data. Running the individual player update model on previous game-play data provides insights into how the model works in the actual game environment. This trait requires the use of a system that automatically generates

responses to presented states using a player model is already established.

First, Algorithm 2 runs with $\delta = 1$ for all state clusters, training the player model on some subset of a player's game-play data (training set). Then it iterates through the remaining game-play instances (test set) and generate a response to each presented state, using both the individual player model and the generic player model. This repeats for each individual player in the game-play dataset. For each test set state, we then determine which response was most similar to the player's actual response. Each time the individual player model is closer than the generic player model to the actual player response, tally a 'win' for the given state cluster and a 'loss' otherwise. The ratio of wins to losses for each state cluster makes up the previous modeling utility trait. The previous modeling utility trait weight, δ_{pma}, is set according to a previous modeling utility threshold. All state clusters with a previous modeling utility above the threshold receive a weight of $\delta_{pma} = 1$ and all others receive a weight of $\delta_{pma} = 0$.

Calculating δ: When Algorithm 2 runs, δ is set to the sum of all trait weights for the given state cluster multiplied by some value q which is an experimental update increment set by the player. Line 6 shows how δ is calculated as a sum of the previously discussed trait weights.

3.4 Case Study: *Space Navigator*

This section demonstrates the player modeling paradigm, focusing specifically on the response generation section of the player modeling paradigm (unshaded area 3 of Figure 12), with a specific application in generating trajectory responses in *Space Navigator*. First, an outline of the initial data capture experiment within the *Space Navigator* environment is presented. Then, solutions are presented to three challenges specific to the game environment: developing a state representation,

comparing disparate trajectories, and finding a trajectory distance measure that is meaningful to humans. These solutions are then used to develop a trajectory response generation algorithm that utilizes a player model to generate trajectories similar to those that would have been provided by an individual player in the same situation.

Initial Data Capture Experiment.

An initial experiment captured a corpus of game-play data for further comparison and benchmarking of human *Space Navigator* game-play. Player data collection used a set of Samsung ATIV Smart PC tablet computers running the Windows 8 operating system. Data was collected from 32 participants playing 16 five-minute instances of *Space Navigator*. The instances represented four difficulty combinations, with two specific settings changing: (1) the number of NFZs and (2) the rate at which new ships appear.

The environment captures data associated with the game state whenever the player draws a trajectory. The data includes: time stamp, current score, ship spawn rate, NFZ move rate, bonus spawn interval, bonus info (number of bonuses, location, and lifespan of each), NFZ info (number of NFZs, location, and lifespan of each), other ship info (number of other ships, ship ID number, location, orientation, trajectory points, and lifespan of each), destination planet location, selected ship info (current ship's location, ship ID number, orientation, lifespan, and time to draw the trajectory), and selected ship's trajectory points. The final collected dataset consists of 63,030 instances, with each player's dataset including an average of 1,950 state-trajectory instances.

State Representation.

Space Navigator states are dynamic both in number and location of objects. Bonuses and spaceships appear and disappear throughout the game and spaceships and NFZs move throughout the scene over time. The resulting infinite number of configurations makes individual state identification difficult. To shrink the large feature vectors obtained in the data capture, the state representation contains only the elements of a state that directly affect a player's score (other ships, bonuses, and NFZs) scaled to a uniform size along with a feature indicating the relative length of the spaceship's original distance from its destination. Algorithm 3 describes the state-space feature vector creation process.

Algorithm 3 State-space feature vector creation algorithm.

input:
L = the straight-line trajectory from the spaceship to its destination planet.

initialize:
$\eta \in [0.0 \cdots 1.0)$ = a weighting variable
s = an empty array of length 19
$zoneCount = 1$

Translate all objects equally s.t. the selected spaceship is located at the origin.
Rotate all objects in state-space s.t. L lies along the X-axis.
Scale state-space s.t. L lies along the line segment from $(0, 0)$ to $(1, 0)$.
for each object type $\vartheta \in (OtherShip, Bonus, NFZ)$ **do**
    for each zone $z = 1 \rightarrow 6$ **do**
        $zoneCount = zoneCount + 1$
        for each object o of type ϑ in zone z **do**
            d_o = the shortest distance of o from L
            $w_o = e^{-(\eta \cdot d_o)^2}$ ▷ Gaussian weight function
            $s\,[zoneCount] = s\,[zoneCount] + w_o$
        end for
    end for
end for
$s\,[19]$ = the non-transformed straight-line trajectory length
Normalize values of s between $[0, 1]$
return s

The algorithm first transforms the state-space features to a straight-line trajectory frame in Line 2. Line 7 translates the state space so the selected ship is at the origin. Line 8 rotates all the objects in state-space so that the straight-line trajectory between the ship and the destination planet is located on the X-axis. Then, Line 9 scales the state-space such that all straight-line trajectories are of equal length. These transformations allow disparate trajectories to be compared in the state-space.

The loop beginning on Line 10 accounts for the different element types and the loop beginning on Line 11 divides the state-space into six zones as shown in Figure 13. The first dividing line creates two zones along the straight-line trajectory. The second and third dividing lines occur perpendicular to the straight-line trajectory at the location of the spaceship and destination planet respectively. This effectively divides the state-space into three zones with relation to the spaceship's straight-line path: behind the spaceship, along the path, and beyond the destination.

To compare disparate numbers of objects, the loop beginning in Line 13 uses a method similar to that used in [161]. Each zone collects a weight score (s) for each object within the zone. This weight score is calculated using a Gaussian weighting function based on the minimum distance an object is from the straight-line trajectory. For objects beyond the destination planet or behind the spaceship, the minimum distance will not be perpendicular to the straight-line trajectory.

Figure 13 shows the transformation of the state into a feature vector using Algorithm 3. The state-space is transformed in relation to the straight-line trajectory, and a value is assigned to each "entity type + zone" pair accordingly. For example, Zone 1 has a bonus value of 0.11 and other ship and NFZ values of 0.00, since it only contains one bonus. The weighting function is evident in the fact that closer entities (Zone 6 - NFZ) have a higher score than entities that are farther away from the straight-line trajectory (Zone 1 - bonus).

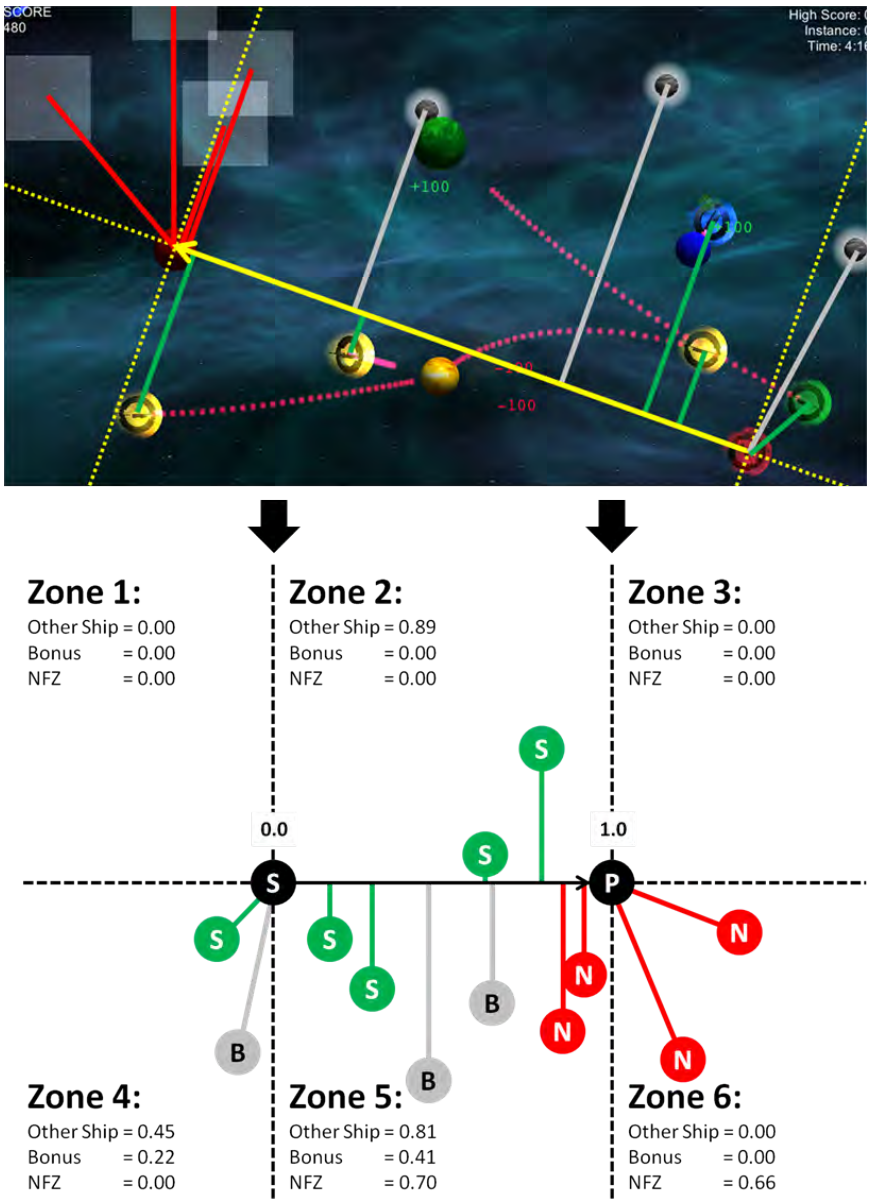

Figure 13. The six zones surrounding the straight line trajectory in a *Space Navigator* state representation and the state representation calculated with Algorithm 3.

Lastly, the straight-line trajectory distance is captured. This accounts for the different tactics used when ships are at different distances from their destination. Ships that are very close to their destination are more likely to result in responses close to a straight-line trajectory, while those that must traverse nearly the entire screen will see a wider variance from the straight-line trajectory. The resulting state

representation values are normalized between zero and one.

Trajectory Comparison.

Trajectory generation requires a method to compare disparate trajectories. This is crucial to being able to determine the similarity or dissimilarity of two response trajectories [166]. However, trajectories generated within *Space Navigator* can vary in composition, containing differing numbers of points and point locations. This section describes how the trajectory generator permits trajectory comparison. Trajectory comparison requires both re-sampling and transformation. Trajectory re-sampling, based on linear interpolation [167], ensures all trajectories consist of the same number of points. The trajectory generator can then compare trajectories using a simpler distance measure.

Algorithm 4 performs trajectory re-sampling. The algorithm begins by keeping the same start and end points, then iterates through until the re-sampled trajectory is filled. The process first finds, in Line 10, the proportional relative position (p_m) of a point. The proportional relative position indicates where the i-th point would have fallen in the original trajectory and may fall somewhere between two points. Calculated in Line 16, the proportional distance (d_m) that p_m falls from the previous point in the old trajectory (p_0) is the relative distance that the i-th re-sampled point falls from the previous point. To compare trajectories, the target number of points is set to 50 for re-sampling all the trajectories (i.e. n_{avg}). Fifty is approximately the mean number of points found in all the trajectories during the initial data capture.

Re-sampling the points in this manner has two advantages. First, the re-sampling process remains the same for both trajectories that are too long and too short. Secondly, the re-sampling process maintains the distribution of points along the trajectory. A long or short distance between two consecutive points, relative to other

Algorithm 4 Trajectory re-sampling algorithm.

inputs:
n_{old} = Number of points raw trajectory we are re-sampling contains
n_r = Number of points to which we are re-sampling
t_{old} = Array of (x, y) points representing the raw trajectory we are re-sampling

initialize:
t_r = Empty array of (x, y) points of length n_r to hold the re-sampled trajectory

$t_r[1] = t_{old}[1]$
$t_r[n_r] = t_{old}[n_{old}]$
for $i = 2 \rightarrow n_r - 1$ **do**
    $p_m = \left(\frac{n_{old}}{n_r}\right) \cdot i$
    $p_0 = \lfloor p_m \rfloor$ ▷ The position directly before p_m
    $p_1 = \lceil p_m \rceil$ ▷ The position directly after p_m
    if $p_m = p_0$ **then**
        $t_r[i] = t_{old}[p_m]$
    else
        $d_m = p_m - p_0$
        $(x_0, y_0) = t_{old}[p_0]$
        $(x_1, y_1) = t_{old}[p_1]$
        $t_r[i] = (x_0 + d_m (x_1 - x_0), y_0 + d_m (y_1 - y_0))$
    end if
end for
return t_r

consecutive point distances within the trajectory, remains in the re-sampled trajectory. This ensures that trajectories drawn quickly or slowly maintain those sampling characteristics to some extent.

Once re-sampled, trajectories are translated, rotated, and resized in relation to the straight-line trajectory. Since *Space Navigator* state-space feature vector creation geometrically transforms a state, the trajectories generated in response to the state must be transformed in the same manner. This transformation ensures the state-space and trajectory response are positioned in the same state space.

Distance Measure.

To ensure the trajectories generated in *Space Navigator* are similar to those of an individual player, a distance measure must capture the objective elements of trajectory similarity such as comparing specific points. Additionally, an ideal similarity measure will also be meaningful to human players, in that the distance measure will be small when a human would think two trajectories are similar and large when two trajectories are dissimilar. A human-subject study confirmed that Euclidean trajectory distance not only distinguished between trajectories computationally, but also according to human conceptions of trajectory similarity. The experiment was conducted as follows in the *Space Navigator* environment.

Each of the 35 participants played two five-minute games of *Space Navigator* for familiarization purposes. Then each player completed 60 pre-scripted instances taken from previously captured games of *Space Navigator*. Each scenario starts from a paused *Space Navigator* instance and the spaceship upon which the player is expected to act blinks. The player responds to the scenario by drawing a trajectory for the blinking ship. The game is paused and the trajectory response is recorded. The scenario is then shown to the player again, with their trajectory replaced by three new trajectories superimposed onto the state. The player is asked to choose the trajectory that is "most similar" to the one they drew.

The three trajectories shown to the player include a straight line from the spaceship to its destination planet, the trajectory in the game-play database that is closest to the provided trajectory according to Euclidean trajectory distance, and the response trajectory to a random state from the same state cluster as the current state. The trajectories are presented as A, B, and C in randomized order. The trajectory selected by the player is recorded as the player's choice as the most similar trajectory. The final collected dataset consists of 35 players completing 60 instances each, for a

total of 2,100 instances.

Average intra-trajectory distance between all three presented trajectories and trajectory length show Euclidean trajectory distance's effectiveness. If Euclidean trajectory distance properly captures human conception of trajectory similarity, small intra-trajectory distances should mean that all three trajectories are similarly indistinguishable to humans. Very small intra-trajectory distances should indicate an almost random choice of "most similar" trajectory for the player, while the smallest Euclidean trajectory distance from the trajectory the player drew to one of the presented trajectories should be chosen with regularity at high average intra-trajectory distances. Additionally, smaller straight-line trajectory lengths allow for less distinguishability due to the constrained nature of possible actions at shorter distances. Therefore, small trajectory lengths should induce less certainty in the choice of "most similar" trajectories.

The results in Figure 14 show that Euclidean trajectory distance captures human conception of trajectory similarity well. All histograms were compiled in MATLAB, the number of bins (k) set according to Rices rule [168] ($k = 2n^{1/3}$, n = the number of observations), and the k bins equally sized between the minimum and maximum trajectory lengths. As expected, those trajectories presented with extremely small average intra-trajectory distances are chosen at an essentially random rate (23.8%). As the average intra-trajectory distance grows, the shortest Euclidean trajectory distance aligns with human conceptions of "most similar" at rates approaching 100%.

Euclidean trajectory distance also accounts for humans being less able to distinguish between shorter trajectories. Since players are more constrained in possible trajectory choices at short straight-line trajectory lengths, the average intra-trajectory distance correlates well with length as demonstrated in Figure 15. This shows a strong

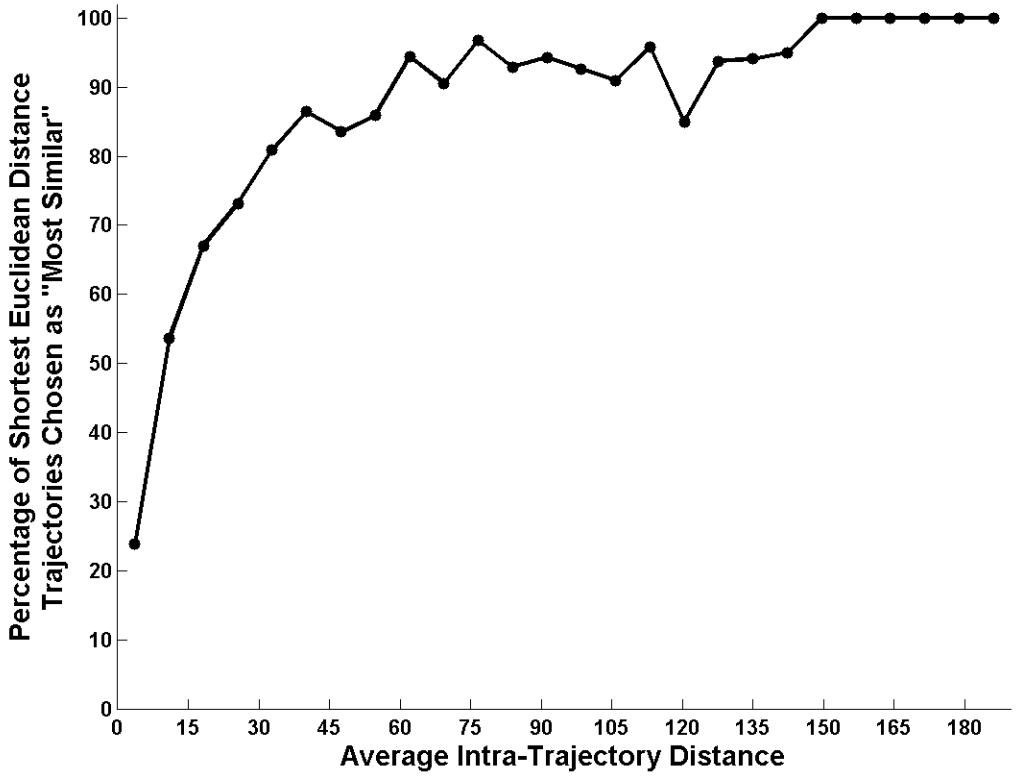

Figure 14. The percentage of times human conception of "most similar" trajectory agreed with the trajectory deemed most similar according to Euclidean trajectory distance as a function of the average intra-trajectory distance.

positive correlation ($r = 0.5635$, $p = 0$).

Combining these insights, Figure 16 shows as trajectory length increases, the percentage of trajectories classified as most similar by humans more regularly matches with Euclidean trajectory distance. Euclidean trajectory distance, therefore, serves as an adequate measure of trajectory similarity in the *Space Navigator* game.

Generate Response.

The response generator utilizes a player model $\mathbf{P}$ to generate player responses. This section describes a method to generate new trajectory responses using the cluster weights in $\mathbf{P}$ that derive from either a generic or learned player model.

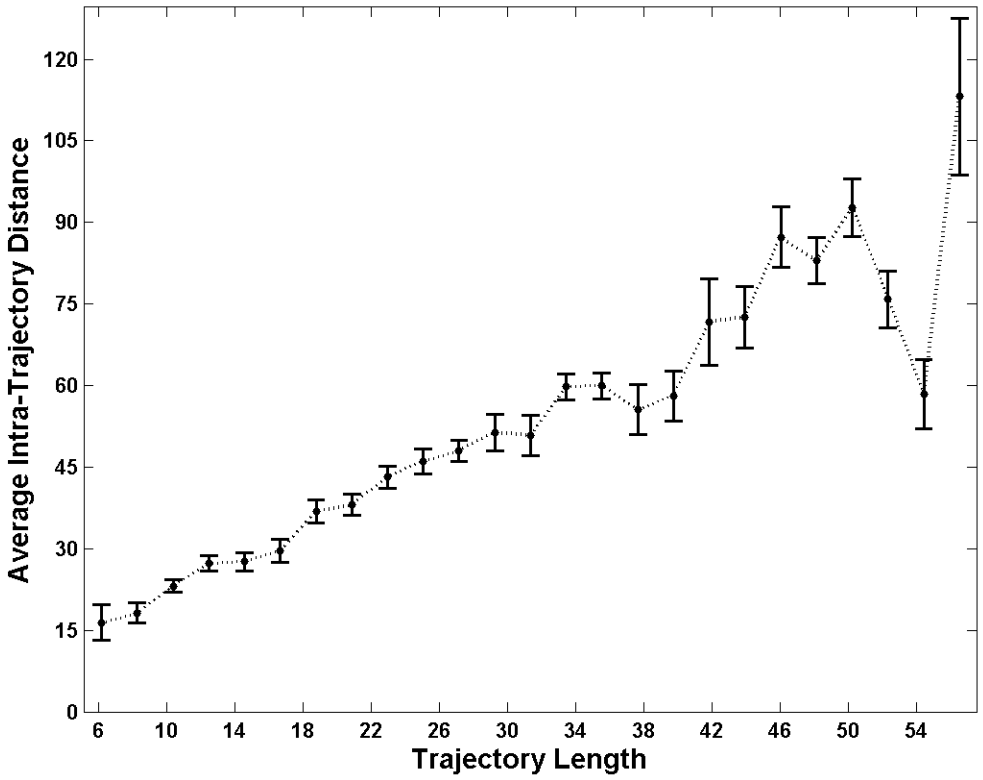

Figure 15. Average intra-trajectory distance as a function of trajectory length.

Existing trajectory generation research has tended to gravitate toward methods creating trajectories one point at a time. Using methods like trajectory libraries [169] or Gaussian mixture models [170], the trajectory generator predicts only the next point on the trajectory. Then it recursively continues the process of creating further points until it reaches the desired en state and returns the entire created trajectory. However, humans tend to think in terms of "full maneuvers" when generating trajectories, specifically for very quick trajectory generation tasks such as trajectory creation games [165]. Therefore, the *Space Navigator* trajectory response generator creates "full maneuver" trajectories.

The trajectory response generation algorithm takes as input: the number of trajectories to weight and combine for each response (k), the number of state and trajectory

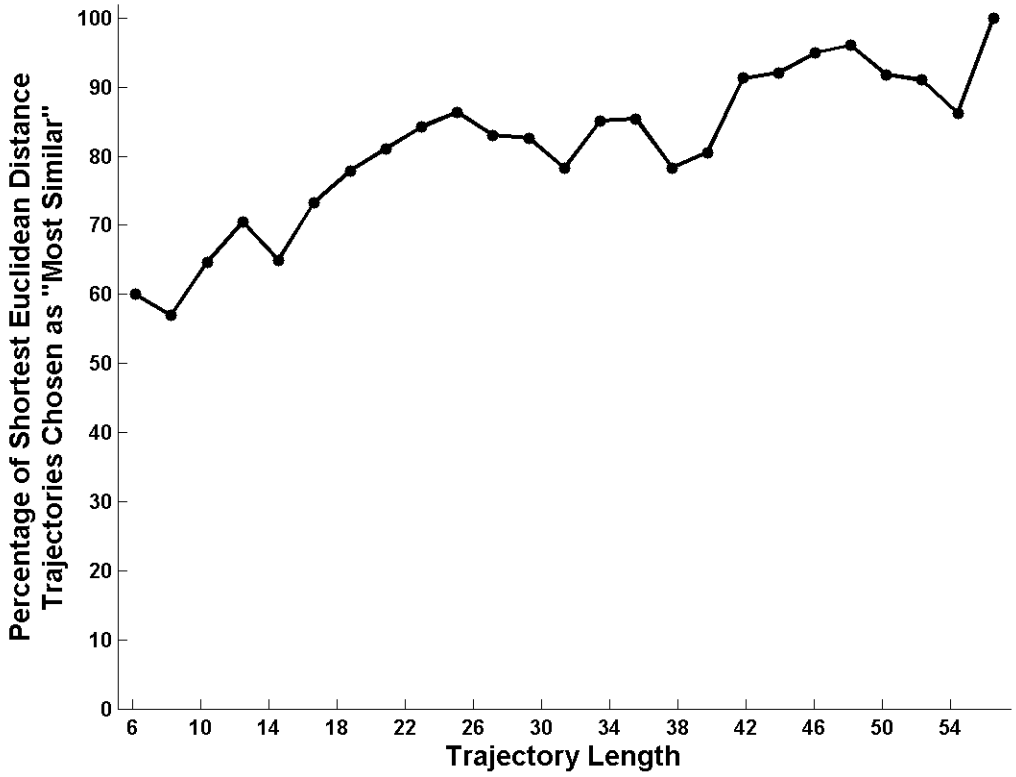

Figure 16. The percentage of times human conception of "most similar" trajectory agreed with the trajectory deemed most similar according to Euclidean trajectory distance as a function of trajectory length.

clusters (x and y respectively), the re-sampled trajectory size (μ), a new state (s_{new}), a player model ($\mathbf{P}$), the set of all state-trajectory cluster mappings ($\mathbf{M}$).

Line 10 begins by creating an empty trajectory of length μ which will hold the trajectory generator's response to s_{new}. Line 11 then finds the state cluster (S_{close}) to which s_{new} maps. $\mathbf{P}_{close}$, created in Line 12, contains a set of likelihoods. $\mathbf{P}_{close}$ holds the likelihoods of the k most likely trajectory clusters to which state cluster S_{close} maps.

The loop beginning in Line 13 then builds the trajectory response to s_{new}. Line 15 finds the instance assigned to both state cluster S_{close} and trajectory cluster T_i with the state closest to s_{new}. The response to this state is then weighted according to

Algorithm 5 Trajectory response generation algorithm.

1: **inputs:**
2: k = the number of trajectories to combine
3: x = the number of state clusters
4: y = the number of trajectory clusters
5: μ = the re-sampled trajectory size
6: s_{new} = a state we have not seen before
7: $\mathbf{P}$ = an $x \times y$ player model
8: $\mathbf{M} = \{\langle S_1, T_1\rangle, \langle S_1, T_2\rangle, \cdots, \langle S_x, T_y\rangle\}$, all state-trajectory cluster mappings

9: **initialize:**
10: $t_{new}(\mu) \leftarrow$ an empty trajectory of μ points

11: S_{close} = the closest state cluster to state s_{new}
12: $\mathbf{P}_{close} = \max_k \left[\mathbf{P}_{S_{close},(z|\forall z \in 1,\ldots,y)}\right]$
13: **for** each $P_{close,i} \in \mathbf{P}_{close}$ **do**
14: $\quad T_i$ = the trajectory cluster associated with $P_{close,i}$
15: $\quad s_{close,i} \leftarrow$ state closest to s_{new} in $\langle S_{close}, T_i\rangle$
16: $\quad t_{close,i} \leftarrow$ the response trajectory to $s_{close,i}$
17: $\quad$ **for** $\nu = 1 \rightarrow \mu$ **do**
18: $\quad\quad t_{new}(\nu) = t_{new}(\nu) + t_{close,i}(\nu) \cdot P_{close,i}$
19: $\quad$ **end for**
20: **end for**
21: $t_{new} = t_{new} / \sum_{i=1}^{k} P_{close,i}$
22: **return** t_{new}

the likelihoods in $\mathbf{P}$. The loop in Line 17, then combines the k trajectories using a weighted average for each of the μ points of the trajectory. The weighted average trajectory points are then normalized across the k weights used for the trajectory combination in Line 21. The trajectory returned by Line 22 is the trajectory response generation algorithms response to state s_{new} according to the player model $\mathbf{P}$

3.5 Experiment and Results

This section describes an experiment to test the real-time individual player modeling trajectory generator and presents insights gained from the experiment. The

results show that the individual player modeling trajectory generator is able to create trajectories more similar to those of a given player than a generic player-modeling trajectory generator, with a limited amount of training data. Additionally, the results show how the model provides insights for a better understanding of what separates different players' game-play through an analysis of the individual player models in comparison to the generic player model.

Experiment Settings.

The experiment compares trajectories created with the generic player model and the individual player model, they are further compared with a trajectory generator that always draws a straight line between the spaceship and its destination planet. The first five games worth of state-trajectory pairs are set aside as a training dataset and eleven games of state-trajectory pairs are set aside as a testing dataset, with each game containing on average 123 state-trajectory pairs. Five training games was chosen as a benchmark for learning an individual player model to force the system to quickly pull insights that would manifest in later game-play. For each of 32 players, the individual player model is trained on the five-game training dataset using Algorithm 2 with the trait score weights. The generic player model and straight-line methods do not require training.

Next, each state in the given player's testing set is presented to all three trajectory generators. The difference between the generated trajectory and the actual trajectory provided by the given player is recorded. The experiment presents states in the order recorded in the original games. The individual player model does not train on the testing data. The experiment also saves the individual player models for later comparison and evaluation. Table 1 shows specific experimental values for the individual player model.

Table 1. Experimental variable settings for individual player modeling using Algorithm 2

Variable	Value
Update Increment (q)	0.01
Cluster Population Threshold	240
Cluster Mapping Variance Threshold	17.0
Previous Modeling Utility Threshold	3.0

The three learning thresholds were set specifically for *Space Navigator* as follows. The state cluster population threshold is set at a value of one standard deviation over the mean cluster size, specifically 240. Forty of 500 state clusters received a cluster population weight of $\delta_{cp} = 1$ and 460 received a population weight of $\delta_{cp} = 0$. The cluster variance ratio threshold is 17, with 461 of 500 state clusters receiving a cluster variance weight of $\delta_{cmv} = 1$. For the previous modeling utility, a player model was trained for each of the 32 players with five games worth of data. Then each of the remaining 11 games were predicted using both the trained player model and the generic player model. For each state across all 32 players, a Euclidean trajectory distance from the generic and individual player models predicted trajectories was calculated from the actual trajectory responses. The cutoff is a learning value of 3, with 442 of 500 clusters receiving a previous modeling utility score of $\delta_{pma} = 1$.

To account for the indistinguishability of shorter trajectories described in Section 3.4, results were removed for state-trajectory pairs with straight-line trajectory length less than length 10.12 meters in the *Space Navigator* environment (approximately 3.5 centimeters on the tablets with 29.5 centimeter screens used for experiments). This distance was chosen as it represents the intersection in Figure 16 at which trajectory lengths reach an accuracy one standard deviation below the mean of trajectory similarity classification accuracy.

Individual Player Modeling Results.

Testing of the game-play databases shows that the trajectories generated using the individual player model significantly improved individual player imitation results when compared to those generated by the generic player model and the straight line trajectory generator. Table 2 and Figure 17 show results comparing trajectories generated using each database with the actual trajectory provided by the player, showing the mean Euclidean trajectory distance and standard error of the mean across all 32 players and instances.

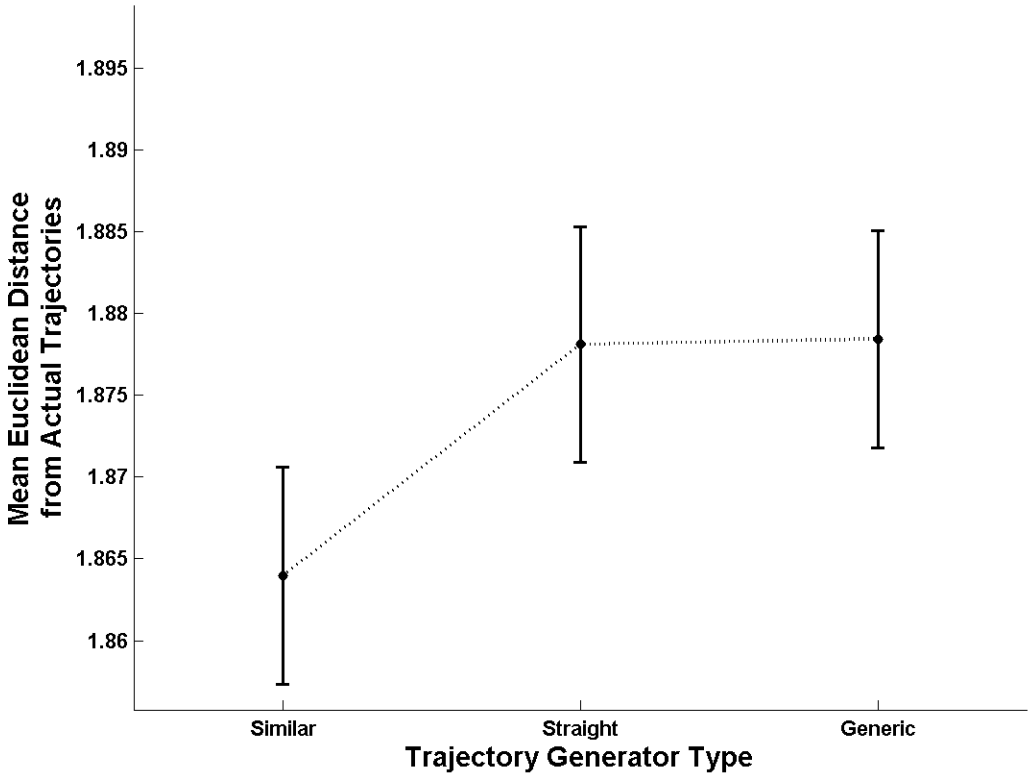

Figure 17. Euclidean trajectory distance between generated trajectories and actual trajectory responses across three trajectory generation methods.

The individual player model generator provides an improvement over the other models. The mean Euclidean trajectory distance of 1.8640 provides a statistically

Table 2. Mean and standard error of the Euclidean trajectory distances (in *SpaceNavigator* environment meters) across all state-trajectory pairs.

Database	Mean Euclidean Traj Dist	Std Err
Individual Player Model	1.8640	±0.0063
Straight Line Generator	1.8781	±0.0069
Generic Player Model	1.8784	±0.0063

significant improvement over the straight line and generic player models, as standard error across all instances from all 32 players does not overlap with the latter two player models. The similar player model improves the generic databases accuracy by learning more from a selected subset of presented states to ensure that the player model more accurately generates similar trajectories.

Individual Player Model Insight Generation.

The individual player models provide insight into general and specific game-play. Comparing the player model learning value changes with the aspects of a state representation allows us to understand what aspects of a state influence game-play and to what degree. How player model changes correlate with the state representation enables game designers a better understanding of what distinguishes individual game-play within the game environment. In turn, this understanding allows for game design improvements.

Table 3 shows the results of a Pearson's linear correlation between the mean learning value change of each state cluster across all 32 players and the state representation values of the associated state cluster centroids. The results show that there is a statistically significant negative correlation between the mean learning value changes and all of the zones, but some changes are much larger than others.

The overall negative correlation arises among object/zone pairs intuitively. High

object/zone pair score imply a large or close presence of an object of the given type, constraining the possible trajectories available to all players. For example, a large presence of other ships in a given zone influences all players to avoid sending trajectories near that area. Therefore, there is more differentiability of player actions available when more freedom of trajectory movement is available.

With the "Ship to Planet Distance" feature, longer distances correlate to less learning value change among player models, with the strongest correlation of all features: r of -0.6434 and p-value < 0.0001. There are several possible explanations for this behavior, including: (1) players are more constrained over long distances and therefore differentiate their actions less, (2) as distances get longer, the variance in the way an individual player draws trajectories in similar situations increases, therefore allowing for no learning of individual tendencies, (3) shorter distances better capture consistent tendencies that a player will carry along to distinguish his game-play over time.

Another aspect that Table 3 begins to show is the importance of the middle zones in comparison to the "before" and "after" zones. Figures 18, 19 and, 20 illustrate this point graphically.

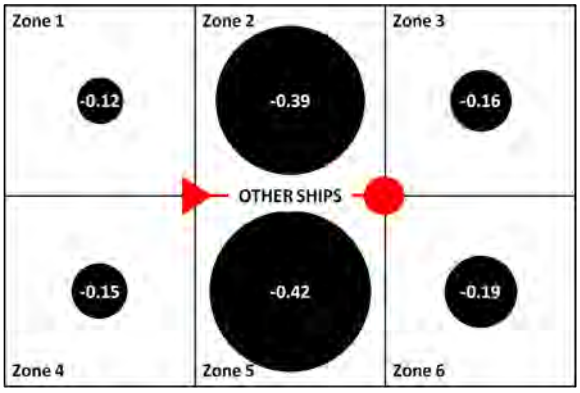

Figure 18. Graphical representation of the correlation coefficient for each Other Ship/-Zone score with the mean change in learning values in player models.

The r values show that the middle two zones provide an larger influence on the

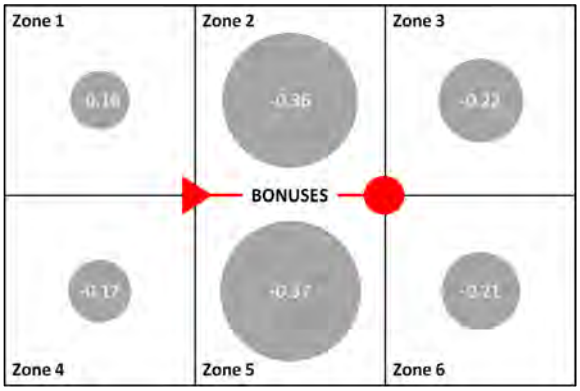

Figure 19. Graphical representation of the correlation coefficient for each Bonus/Zone score with the mean change in learning values in player models.

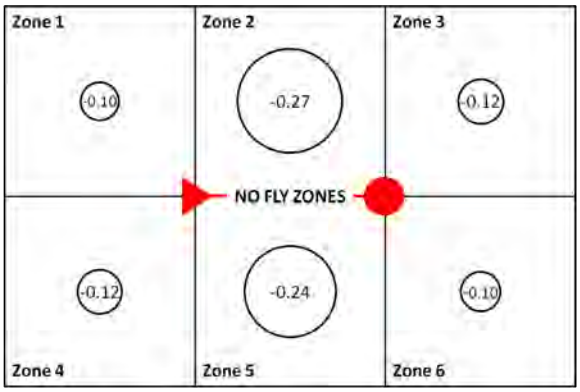

Figure 20. Graphical representation of the correlation coefficient for each No Fly Zone/Zone score with the mean change in learning values in player models.

amount of change in the learning values. For example, in Figure 18 the r values for zones two and five are more than double those of any other zone. This idea is somewhat intuitive as this is the area that the ship will traverse, providing the most likely cause for interaction with objects of any given type.

Figures 18, 19, and 20 and Table 3 provide insight into the relative value that players place on certain types of objects. For example, determining the correlation coefficients of different Object/Zone Pairs can show that No Fly Zones in the middle two zones provide a significantly smaller influence on learning value changes than other ships do in the same zones. Since there is such a large difference, we can infer

that players reactions to other ships are more valuable in determining how a person will play the game than No Fly Zones.

Three examples of how player modeling insights can be used in game applications involve training, game design, and player automation. The player models can be used to find places where specific users who are doing really well are properly valuing certain actions (e.g. avoiding other ships) according to the incentives. Proper valuations can then be communicated to players during training within the environment. Another example is that, we can use the player modeling insights to design point structures to more closely align with the way players perceive the value of different object types. In *Space Navigator*, increasing the point magnitudes of No-Fly Zones and Bonuses makes the game more difficult by equally balancing the incentive structure, encouraging less focus on a single objective over the others. Lastly, modeling a specific player enables the designer to incorporate an automated player to play like a specific expert or current user within the game.

3.6 Conclusions and Future Work

The real-time individual player modeling paradigm presented in this paper is able to generate trajectories similar to those of a specific *Space Navigator* player. The system is able to operate in real-time without needing to perform time-consuming offline calculations to update player models. Additionally, the gains in individual player imitation are found in a relatively small amount of game-play (five games/25 minutes). The player models developed to imitate players also allow for a better understanding of what traits of a given state provide understanding of player differences which occur for different states.

This work provides opportunities for several areas of future work. Further studies will research the effects of using the trajectory generator to act as an automated

aid for players interacting with the *Space Navigator* game. Additionally, further analysis of the player modeling methods could yield further insights into how much differentiation of individual players can be gained over different amounts of time. Moreover, imitating individual players could provide helpful insights in determining how experts play *Space Navigator* to aid in experiments to learn how to improve player training.

Table 3. Correlation of each state representation value with the mean change in associated state cluster learning values in player models

Value	**Pearson's r**	**p-value**
Zone 1 - Other Ships	−0.1227	0.0060
Zone 2 - Other Ships	−0.3911	0.0000
Zone 3 - Other Ships	−0.1616	0.0003
Zone 4 - Other Ships	−0.1465	0.0010
Zone 5 - Other Ships	−0.4244	0.0000
Zone 6 - Other Ships	−0.1903	0.0000
Zone 1 - Bonuses	−0.1569	0.0004
Zone 2 - Bonuses	−0.3552	0.0000
Zone 3 - Bonuses	−0.2212	0.0000
Zone 4 - Bonuses	−0.1662	0.0002
Zone 5 - Bonuses	−0.3693	0.0000
Zone 6 - Bonuses	−0.2056	0.0000
Zone 1 - NFZs	−0.1002	0.0251
Zone 2 - NFZs	−0.2749	0.0000
Zone 3 - NFZs	−0.1184	0.0080
Zone 4 - NFZs	−0.1159	0.0095
Zone 5 - NFZs	−0.2398	0.0000
Zone 6 - NFZs	−0.1040	0.0200
Ship to Planet Distance	−0.6434	0.0000

IV. Adaptive Automation System Design Life Cycle

With a trajectory drawing automation system in place for the *Space Navigator* environment, the focus of research moved to developing an adaptive automation system with it. However, our initial placement of the adaptive automation system within the *Space Navigator* task relationship diagram proved unsuccessful. This setback spurred the generation of a development life-cycle to address the setbacks and characteristics common to adaptive automation system design and implementation.

The lessons learned from the *Space Navigator* adaptive automation design and implementation process are presented in this chapter. As in previous chapters, the work presented here is a slightly modified version of a paper submitted to the *IEEE Transactions on Human Machine Systems*,[1] with repeated material removed.

4.1 Introduction

Automated systems bring the promises of reduced manpower costs and human error, creating systems that will reduce human workload within complex environments. Tasks that were previously performed by a human can theoretically be offloaded onto a machine to achieve workload reductions. Human workload reductions then allow the human to perform increasingly desirable tasks. However, operators can over-rely on automation, decreasing their situation awareness and allowing skills which are necessary during automation failures to atrophy. To address this problem increasing research emphasis has been placed on Adaptive Automation (AA) [4, 63, 90]. Several research efforts have addressed aspects of AA system design [1,3,4,25,42], but do not propose a system development life-cycle to implement AA in systems from beginning to end. To address this problem, this paper presents a novel Adaptive Automation

[1]Bindewald, J. M., Peterson, G. L., Miller, M. E., and Langhals, B. T. An adaptive automation system design life cycle. *IEEE Transactions on Human Machine Systems* (SUBMITTED).

System Development Life Cycle (AASDLC).

The AASDLC improves upon previous systems and software design life-cycles by placing a focus on issues related to adaptive automation. These issues include identifying locations for AA within a system, dealing with unexpected changes from adding AA, and understanding how to trigger AA changes. The AASDLC begins by utilizing the Function to Task Design Process Model (FTTDPM) [1] that aligns the system's design with the actual implementation before AA is added to the system. User testing then enables the practitioner to ensure that the implemented system reflects the actual system in operation. Using the insights gained from user testing allows the FTTDPM to identify useful areas for AA within the existing system. The resulting automation is then implemented using an AA trigger and human-machine interface, designed with the overall AA goal in mind. Another set of user tests ensures that user behavior when employing the AA aligns with expectations and the AA goals in the implemented system. Multiple iterations within segments of the life-cycle enable the creation of a system that will be ready for release.

This research provides three contributions to AA system development. First, it presents the AASDLC created specifically to address AA system design and implementation. The second is an AA user feedback model, that assesses user feedback on two dimensions: 'qualitative vs. quantitative' and 'directed vs. undirected.' This model is used throughout the AASDLC to inform the design and implementation processes. Third, the AASDLC includes a new model for AA triggers that couples Feigh *et al*'s trigger type taxonomy [3] with a trigger mode. The trigger mode enables representation of discrete, continuous, or complex triggers to show the many ways that AA can adapt in a specific situation.

This paper presents the AA system development life-cycle by first explaining the entire process and then stepping through each phase. This life-cycle is then demon-

strated through an example AA system design and implementation in the *Space Navigator* environment, analyzing strengths and weaknesses of the implementation and suggesting improvements through application of the life-cycle.

4.2 Related Work

Research influencing AA system development includes the areas of adaptive automation models, system and software development life-cycles, specialized topic area system development life-cycles, and user centered design methodologies.

Adaptive Automation Models.

Adaptive automation (AA), sometimes known as *adaptive systems*, is the component of a human-machine system that enables dynamic adjustment of the machine portion of the system to the changing environment in which the overall human-machine system operates [1, 3]. AA system design involves implementing both a specific adaptation type and a trigger(s) to adjust the type or level of automation [3].

Adaptation types include changes to function allocation, task scheduling, human-machine interaction, or content; but the most commonly addressed adaptation type is function allocation [3]. Parasuraman *et al.* [4] define an automation allocation as one of ten levels of automation (LOA) across four stages of human information processing. Research has subsequently used dynamic changes in LOA to create AA systems [25, 42]. The FTTDPM for AA system design [1] focuses on adaptations involving function allocation. Although FTTDPM provides a design methodology focusing on where to place adaptive nodes within a system, it does not address how to implement the AA system.

After selecting an adaptation, an AA trigger determines when and to what extent to adjust the automation. Feigh *et al* [3] list five types of triggers: operator based,

system based, environment based, task and mission based, and spatiotemporal. Although work has been done to explore the effects different types and complexities of triggers have on human or system performance, little work has been done on AA trigger design [11, 171].

System and Software Development Life-Cycles.

A *system development life-cycle* is the general term for a structured process for designing and implementing information systems. In many cases the terms system development and software development are used interchangeably when referring to the same life-cycles [172, 173]. Several system development life-cycles have been developed to serve differing purposes. The Waterfall model is a software development life-cycle that follows a sequential step approach through the design process and into implementation [82]. Although useful as an idealized life-cycle, in practice most software development processes are not able to follow such sequential ordering. Therefore, several iterative system development life-cycles [83, 174–177] have been developed to allow simultaneous evolution of requirements and implementation as questionable assumptions are revealed.

The newer life-cycles attempt to decompose aspects of the waterfall model into smaller chunks [174] or create a more agile process where steps can be repeated as the understanding of requirements and system limitations are revealed [176]. As opposed to the waterfall model's assumption that requirements must be completely understood at the start of the process, the new life-cycles allow refinement of requirements throughout the process as the steps accumulate domain knowledge and feedback. Common to all life-cycles are phases that include some form of: requirements gathering, system design, software implementation, software testing, and end product release. Each life-cycle may add or combine steps, change the ordering of

steps, iterate over steps, or add cycles.

A few of the more popular development life-cycles include iterative development [175], the spiral model [83], SCRUM [174], extreme programming [176], and rapid application development [177]. Although several life-cycles exist for systems and software development, AA-specific design considerations are not addressed. AA-specific concerns include identifying possible locations for AA within a human-machine system, handling unexpected changes that result from AA, and developing effective AA triggers. Although different development life-cycles may address some of these concerns, AA implementations must address each concern and potential interaction.

Specialized Topic Area Development Life-Cycles.

Several research areas have created system development life-cycles to address concerns specific to their industry, and their evolution instructs development of AA. One of the more robust of these areas is data mining, where several life-cycles exist. Other areas with specific life-cycles include mixed reality system design and control system software design.

Several data mining system development life-cycles have been proposed, including: Knowledge Discovery in Databases (KDD) [84]; Sample, Explore, Modify, Model and Assess (SEMMA) [178]; Two Crows [179]; Refined Data Mining Process (RDMP) [180]; and CRoss Industry Standard Process for Data Mining (CRISP-DM) [85]. The CRISP-DM process has been the most widely adopted [181], with its two-part nature setting it apart from other life-cycles. Specifically, CRISP-DM defines a data mining process across different levels and then provides a methodology to implement data mining systems.

Another specific application of system development involves human-computer interaction within mixed reality systems (where entities must simultaneously act in

both real and virtual worlds), [182] creates the Extended 2 Tracks Unified Process (2TUP) to address interaction requirements. Di Orio *et al* [86] create a development life-cycle for control systems within the automotive industry considers not only best practices, but also actual systems in place within the industry.

The data mining, mixed reality, and control system software development life-cycles provide examples for the AASDLC, by extending several development principles to a specific focus area. Most of the life-cycles address development as an iterative structure and all of them identify specific concerns that must be addressed due to the peculiarities of the chosen field.

User-Centered Design.

User-Centered Design (UCD), also referred to as User-Centered System Design [110], is "a process focusing on usability throughout the entire development process and further throughout the system life-cycle." The UCD system development life-cycle involves six steps: vision and plan, analyze requirements and user needs, design for usability, evaluate use in context, feedback/plan the next iteration, and construct and deploy. Like several other life-cycles, such as CRISP-DM or the spiral model, the UCD system life-cycle is iterative, with many steps and/or sets of steps repeated as information is gained in the process. To maintain a usability focus, the life-cycle evolves around a set of 12 key principles. Applying these principles, the five most important UCD methods for a successful project according a survey of UCD practitioners [183] include: field studies, user requirements analysis, iterative design, and usability evaluation, and task analysis.

Past research efforts [41,184] have indicated that UCD concepts could improve AA system development. Although UCD may not have a process that explicitly addresses adaptive automation, lessons learned from UCD implementations provide insight into

moving AA systems from the design models that already exist into implementations. These lessons become increasingly apparent with common AA goals such as reducing workload or increasing task throughput.

4.3 Methodology

The Adaptive Automation System Design Life Cycle (AASDLC), shown in Figure 21, creates a methodology to address the design and implementation of adaptive automation. The process consists of six phases, each with component parts. Like the spiral model, CRISP-DM, and UCD processes; the progression through the process is not always linear, especially when expectations of design do not meet the reality of implementation. Further, like the CRISP-DM and 2TUP models, the process is divided into 2 stages, the first focused on the design and implementation of the system before automation and the second on the design and implementation after adaptive automation is added. This two-stage division permits simplification of the design as it reduces the complexity created by trying to design an AA without a complete understanding of the system in place. This section describes each life-cycle phase, comparing the phase to similar stages in existing life-cycles and proposing questions designers should consider.

Phase 1: Define Adaptive Automation Goals.

The first phase in the AASDLC is to define the goals of the system and the AA goal. An *adaptive automation goal* documents **why** the human and machine are interacting within the system in an adaptive manner. AA goals should not define how the human and machine will interact, but rather provide a theoretical reasoning for why adaptive automation is being used in a specific instance. The AA goal should be framed in the context of the overall system's goal.

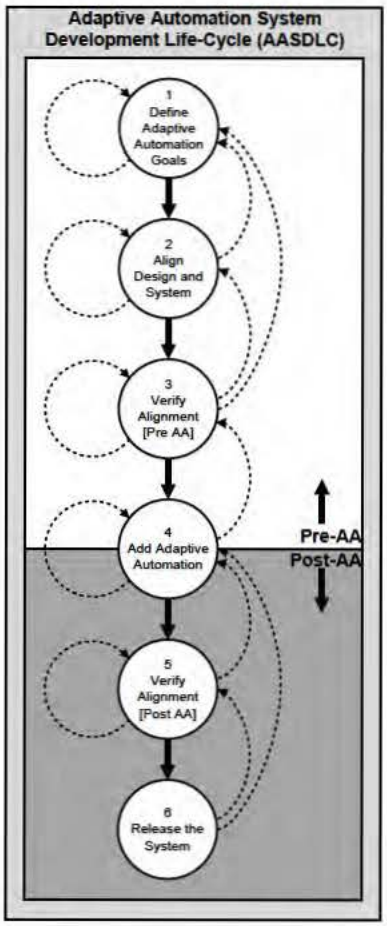

Figure 21. A model showing the progression of phases within the adaptive automation system development life-cycle (AASDLC).

Successfully achieving a well-designed AA goal should improve the chances of the system achieving its goals through the removal of adverse inputs and effects and/or increasing positive inputs and effects. A common AA goal is to reduce human workload within the overall system, while permitting the user to maintain a high level of situation awareness or practice skills which can be important during automation failures [42].

Workload reduction, particularly when the environment introduces elevated workload, increases the system's ability to perform more tasks during periods of excessive task load and allows the human and machine to perform successfully during condi-

tions that would otherwise require multiple people or result in mission failure. This AA goal aims to remove the negative effects of a high human workload and should be as specific as possible, indicating the level of task load to which the system must respond.

The designer should also determine if AA is required to achieve the overall goal, as a system employing AA will be more complex than a nonadaptive system. Further design phases and implementation may change the designer's understanding of the previously defined AA goal's ability to actually help achieve the overall system goal, as indicated by the feedback loops to phase one in Figure 21.

Comparison to Other Life-Cycles.

The 'define AA goals' phase in the AASDLC is similar to the 'Determine Over-Arching Goal' step in the FTTDPM, the 'Vision and plan' phase of UCD and the 'Business Understanding' step of CRISP-DM. The define AA goals phase takes the context of the overall system, and specifically addresses how practitioners intend to use AA to their advantage.

Designer Questions.

- What is the overall system goal? How can AA help achieve this goal? How could AA hurt this goal?
- Why is AA required? What do we want the AA system to adapt to?
- Is AA required to achieve this goal? Is there a simpler system that achieves this goal without adaptability?

Phase 2: Align the design and system.

As shown in Figure 22, aligning the design and the system is a cyclical two-fold process involving **capturing** the system in the design and **implementing** the design in the system. To *capture the system as implemented* one must capture every system task, whether the functions are instantiated as human tasks or machine tasks. An accurate human-machine system design provides better insight into how AA can be added. While this phase appears to assume that an existing technical system exists, this is not required but might include any system(s), technical or procedural, which accomplishes the goals of the system under design. Two tasks are needed to *implement the design in a system.* First, the designer must ensure all functions represented in the design are performed in the system. Second, the designer must ensure that each function is instantiated properly as a human or machine task according to design specifications.

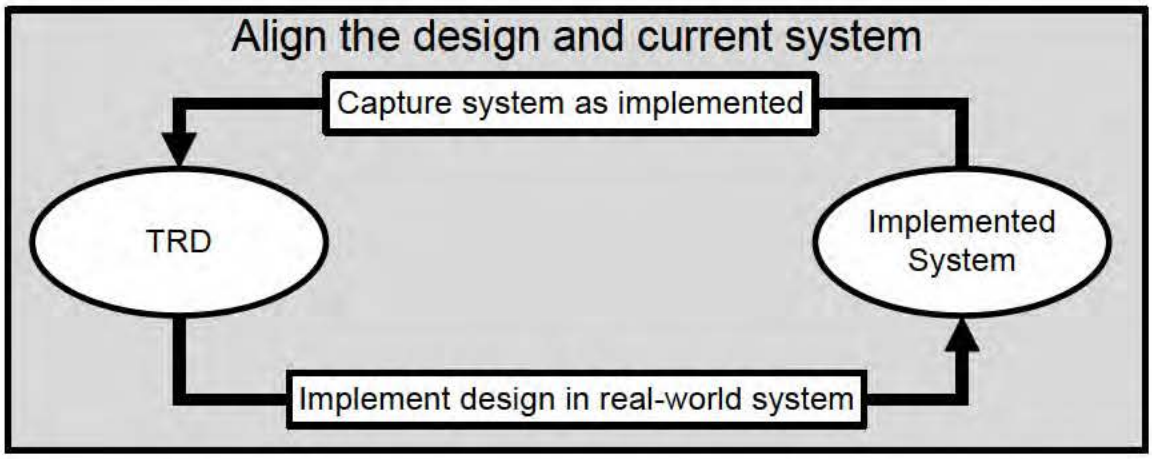

Figure 22. The align design and system phase of the AASDLC, ensures the system implements the design and the design properly represents the system.

A description methodology must be used to design a new system or capture the existing one. Several description methodologies exist, including ConcurTaskTrees [124], DIANE+ [122], GOMS [121], and HAMSTERS [129]. However, the FTTDPM [1] was

designed specifically to help determine where AA could fit well within a system. As such, the Function Relationship Diagrams (FRD) and Task Relationship Diagrams (TRD) of the FTTDPM form the underpinnings of the AASDLC. The FRD captures all of the functions performed within a designed system, their temporal dependence, and the information which must be passed from one task to enable the subsequent task. The TRD then instantiates each function either to a human or a machine. Using the concept of *inherent tasks*, which include functionalities not present when a function is unallocated, but arise only when the function is assigned to a human or machine entity (e.g. a human-machine interaction task that arises when function control moves from a human to a machine), the TRD then allows the practitioner to determine where AA could benefit the overall system by considering several factors including the information that must be conveyed between the human and system to facilitate the handover of tasks between the two entities. The end product of the FTTDPM, the Auto-TRD, is a TRD that represents AA nodes within the design.

Capturing the system involves steps 2-6 of the FTTDPM, 'identify high-level functions,' 'decompose functions,' 'construct function relationship diagram,' 'instantiate functions to tasks,' and 'separate inherent tasks' respectively.

Additionally, in the case where there is no pre-existing system, implementing the system as currently designed becomes paramount. The main reason for this is to help determine the limits of the automation. Without implementing the TRD into an active system, an AA could be designed that is not feasible. This phase involves several iterations of the process in Figure 22–adjusting the TRD and changing system implementation until the two align. To move beyond this phase, the system implementation need not be to the level of an end user system.

Comparison to Other Life-Cycles.

Capturing the system design within the TRD overlaps with a few different life-cycles in different ways. Within UCD the 'vision and plan' and 'design for usability' steps are important in capturing the system design, but it could overlap with other steps as well. Within the CRISP-DM framework, the *capture system as implemented* portion of this phase aligns under 'business understanding.' On the other hand, the *implement design in system* portion of this phase falls in disparate steps: aligning well with 'prototyping' in the spiral model and the 'data understanding' phase in CRISP-DM.

Designer Questions.

- Which functionality in the system is allocated as a human task and which as machine?
- Who would better perform each function in the system, the human or the machine?
- What inherent functionalities (extra tasks) are present that may not be captured in the initial designs?
- Will automation be feasible, timely, and effective for the items allocated to the machine?

Phase 3: Verify design and implementation alignment through user testing.

The next phase of the AASDLC utilizes user testing and feedback to **verify** design and system alignment, prior to AA inclusion. User feedback designed along two dimensions, quantitative vs. qualitative and directed vs. undirected, allow for both

inductive and deductive insights of how well the system meets AA goals. Figure 23 shows the phase in its entirety.

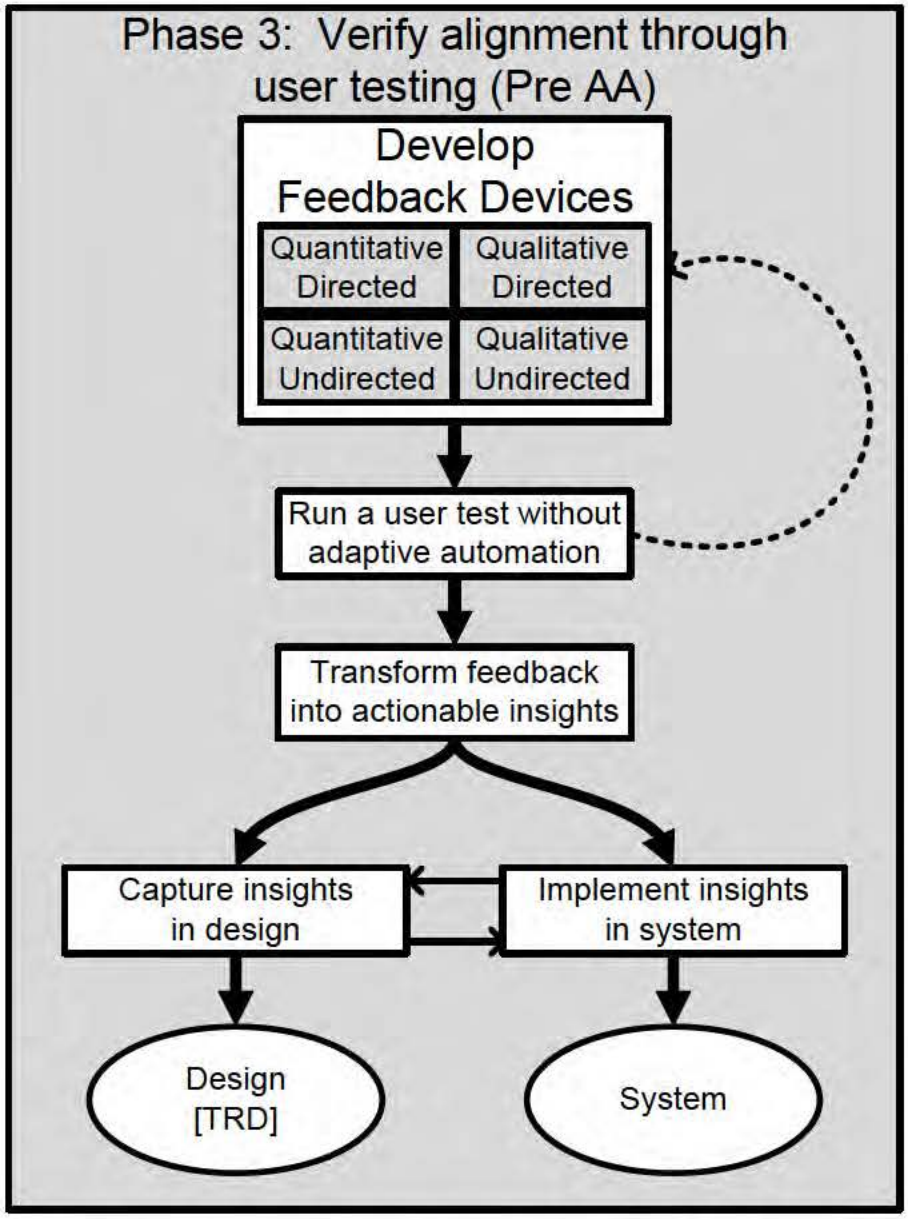

Figure 23. The verify alignment through user testing (Pre AA) phase of the AASDLC, gathers feedback from users to enable to a better understanding of the system before AA is added.

Feedback devices take one of two forms, quantitative or qualitative. Quantitative feedback is numerical and allows the practitioner to compare responses mathemati-

cally, while qualitative feedback allows the user to elaborate on a topic of interest– which can make comparison difficult. The purpose of obtaining both forms of feedback is that quantitative feedback empowers deductive reasoning, while qualitative feedback feeds inductive reasoning [185]. Quantitative feedback can be used deductively to prove or disprove an already held assumption. As such, quantitative feedback helps us begin at the goal and find user data to either prove or disprove whether the overall system goals are met in the system. Qualitative feedback can be used inductively to build the system's overall goal from user experiences. We can use qualitative feedback from user observations to make broader assumptions about how the system works in practice, thus deciphering what the users see as the goal of the system, apart from any preset notions of the goal.

Feedback is then gathered either in a directed (e.g. prompting feedback on specific items) or undirected manner (e.g. asking for general feelings on the system). Directed feedback can be used to ensure that feedback is gathered surrounding a specific interest item. Directed feedback can give insight into the design itself or even directly address whether the goal is met. Undirected feedback, on the other hand allows for unearthing more general sentiments or beliefs about the system that would not be readily apparent.

Example feedback devices from each of the four quadrants, shown in Figure 23 include: Asking each user to verify the system TRD by adding or removing items would be a form qualitative directed feedback. A quantitative directed feedback mechanism could be to ask each user to rate the difficulty or importance of each task within the system TRD. A qualitative undirected feedback mechanism for helping with AA design could include asking for sub-tasks within the system that would be useful to have the computer perform intermittently. Quantitative undirected feedback measures must be collected within the system without directing the user to a specific

focus, requiring the system to record user performance data within the environment such as time on task or performance score.

A collection of all four types of feedback provides a more robust design and implementation alignment, but care must be taken to focus feedback on the AA goal. If the goal of AA is to relieve the user of stress at all costs, feedback geared toward workload should take precedence over feedback determining engagement levels. The last component of feedback device design is to form a prediction of the feedback that will be gathered by each device. Forming this prediction allows the practitioner to have a point of comparison between "how he or she expects the human-machine system to behave" and "how it actually behaves during user tests."

Designing the user test should ensure that a sufficient number of users are sampled, the users sufficiently represent the class of users who will use the end system, and the task environment sufficiently mimics the target task environment. During the user test, feedback devices should be checked to address any unexpected errors. If any devices provide no, incomplete, or unexpected feedback; they can be corrected at an early stage. Improperly designed feedback devices or unexpected feedback can cause a complete repeat of the user test.

Once the user test is complete, the feedback received must be transformed into actionable insights. Transforming feedback into *actionable insights* for verifying design and system alignment begins by comparing the feedback predictions to the user results, and then creating specific changes that can be made to the design or system. Feedback that differs even slightly from expectation indicates a flaw in the practitioner's understanding of how the system behaves, and therefore a problem with alignment. These insights should then be captured in the design and implemented in the system.

Comparison to Other Life-Cycles.

The 'verify design and implementation alignment through user testing' phase of the life-cycle compares well to the 'Feedback' and 'Construct' phases of UCD, giving the practitioner insights that may be missed due to the level of involvement in the design of the system. This phase also imitates the incremental model's approach to deploying a system multiple times, but avoids the need to completely step through the entire process each time.

Designer Questions.

- Does user sentiment agree with our design of the system? If not, where?
- Where do users think AA would benefit the system?
- Does data gathered from user interaction with the system support the flow of information in our design?
- What hidden factors did we not capture in our initial system design documents?

Phase 4: Add adaptive automation to design and system.

The fourth phase of the AASDLC sets it apart from other system development life-cycles. The goal of this phase is to **add AA** to the system, resulting in a system with an integrated AA element, trigger, and interface along with an Auto-TRD that reflects the previous TRD with AA nodes identified. First, this phase adds an automation element. Then, an AA trigger is designed that adaptively changes between levels of automation. Finally, the design of how the handoff will be performed as automation is engaged (i.e. the human-machine interface) is crucial in that situational information must be effectively communicated between entities. These three stages are highly

interconnected and influence each other. The following paragraphs address each of the stages illustrated in Figure 24.

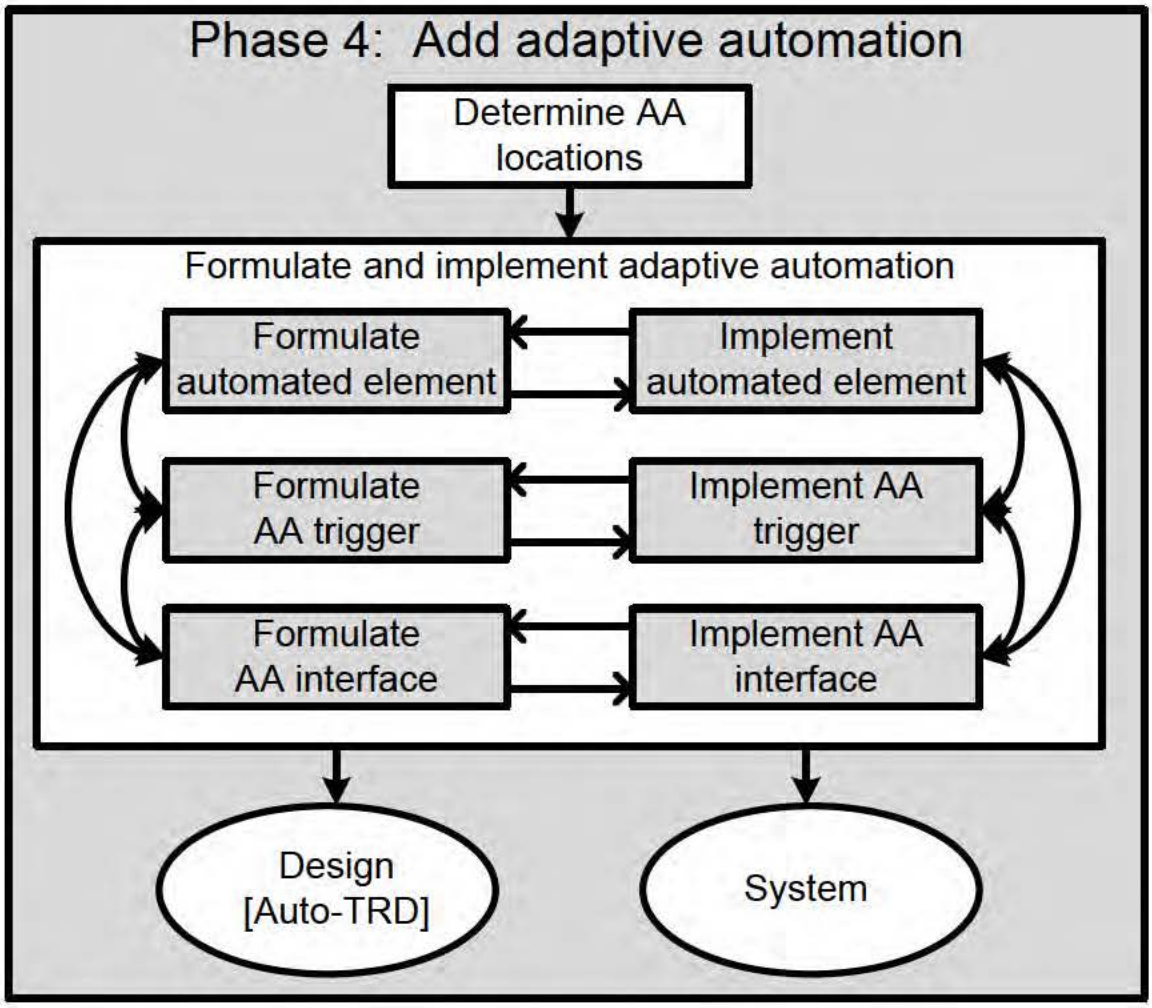

Figure 24. The add adaptive automation phase of the AASDLC creates the AA, which consists of an automated element, AA trigger, and AA interface.

A decision must be made as to what portion of the overall system's task will be automated. This process has been previously outlined in step seven of the FTTDPM, 'Define Adaptive Automation States.' In this method, the designer considered the number of possible automation states to be applied (adapted among), the difficulty or complexity of task handoffs, node clustering of tasks into a unit to be automated, branch counting (e.g. counting the number of pieces of information that must be

communicated between the human and machine entities), and comparing the task load imposed by each task or cluster of tasks which might be potentially automated). Results from previous phases of the AASDLC can help influence the choice of an AA location. Once a location is identified, the next three stages will follow a rough order, but will overlap and create cycles in their design and implementation.

The first stage of AA formulation and implementation is to formulate and implement the automation element. It is important to only automate one portion of the task at a time. If there are multiple elements of the task that will be automated–switching between many discrete levels or even continuously adjusting some portion of the task–adding the automations one at a time will account for unexpected consequences without having to determine which AA element is responsible, unless there are interaction effects. The actual formulation of the automation element is a software engineering problem, wherein some automated system is engineered to perform a portion of the task.

Adaptive automation triggers will fall into one of several types according to the taxonomy created by Feigh *et al* [3], as shown in Figure 25. Further, all triggers switch between discrete automation settings or adapt the automation in some way on a continuous scale. All adaptive automation triggers can be modeled as a distinct pairing of trigger type (from the taxonomy) and mode (continuous or discrete) and the combination of multiple simple triggers can produce ever more complex trigger designs.

The formulation of the associated human-machine interface begins after completing the AA trigger. When determining how to pass information from the human to the machine and vice versa, note what information about the previous tasks must be communicated between the human and machine, as identified in the TRD. The fact that this transfer of knowledge requires perceptual, cognitive, and motor resources on

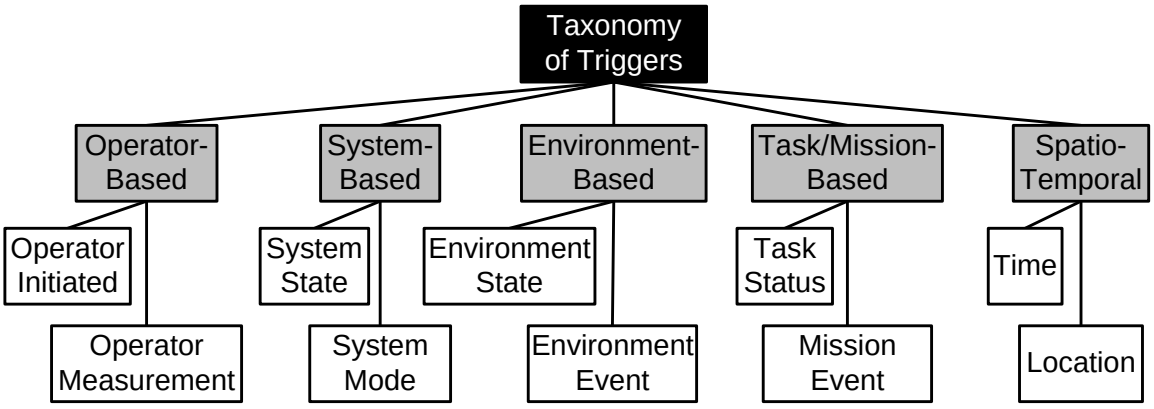

Figure 25. Feigh *et al*'s taxonomy of adaptive automation trigger types adapted from [3].

the part of the human highlights the reason to select the portion of the automated task based upon the complexity and number of elements of information that must be transferred. Insight into the human-machine interface design theory and practice is beyond the scope of this research, and we point the reader to the work of Inagaki [186], Kaber *et al* [41, 71], and Miller *et al* [187].

Comparison to Other Life-Cycles.

Since we are adding AA to an already implemented system, the life-cycle is split into two parts: pre- and post-AA. With this in mind, the most compelling comparison can be made within domain specific system development life-cycles. Most domain specific life-cycles, such as 2TUP, contain similarities to generic system design life-cycles, but add specific process tasks and re-arrange steps to normal methods based on domain insights needed at certain points in the problem. These added steps influence the proceeding process elements. The AASDLC sees some of the same consequences: the post-AA phases of the process imitate the pre-AA phases, but they have differences due to the placement of AA in the system at this specific juncture.

Designer Questions.

- What locations in the TRD prove most conducive to AA based on the FTTDPM AA identification tools?
- How can we automate the chosen element within the task structure? What impacts will the specific chosen method have on the surrounding elements?
- What type of AA trigger should we use based on the overall AA goal? What mode (continuous or discrete) should we use to adjust the automation?
- What information needs to be communicated between entities during AA hand-offs? How can the human-machine interface effectively communicate this information?

Phase 5: Verify AA design and implementation alignment through user testing.

The user test for phase five is similar to the methods used in phase three. The difference is that the feedback should tie specifically to assessing the goal of the AA. The goal of this phase is to **verify AA** designed and implemented reflect each other. Data gathered through user testing should allow the practitioner to determine whether or not the design choices and implementation reflect what real users experience in the system.

In this phase, the FTTDPM's idea of inherent tasks is useful. When interacting with a system that is adaptive, users may discover new functions that are inherently assigned to them that were not considered previously. The arrival of unforeseen inherent tasks should prompt a reiteration of this phase or a complete revisiting of one of the previous phases. Both users who participated in previous system tests without AA implemented and users who've never seen the system before should participate

in this phase. The differences in how the users perform the task can reveal unseen biases in design and implementation.

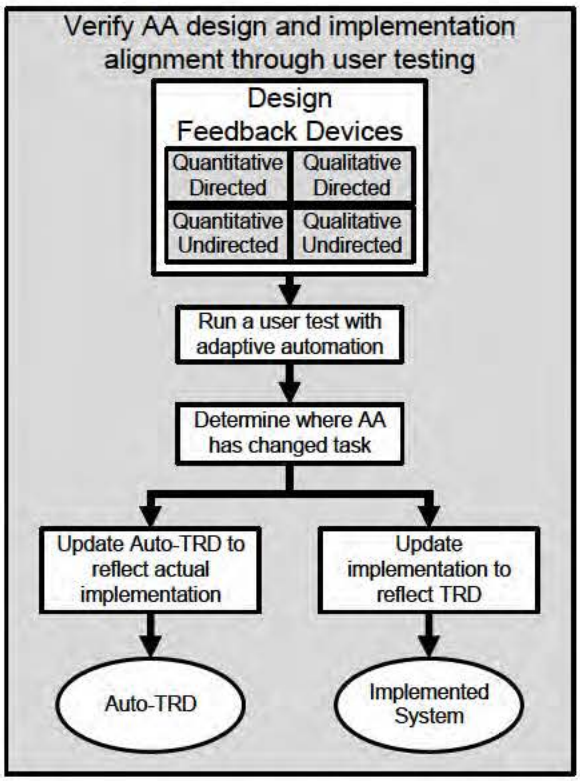

Figure 26. The verify AA alignment through user testing (Post AA) phase of the AASDLC, utilizes user feedback to ensure the system operates as expected after AA is added.

Comparison to Other Life-Cycles.

Similar to phase three, this one is also similar to the 'Feedback' and 'Construct' phases of UCD, specifically helping to illuminate unforeseen consequences of AA.

Designer Questions.

- Does the system as implemented meet its AA goals according to user sentiment and performance?
- Can the users understand the implemented AA? Does it act as expected?

- Does data gathered from user interaction with the system support the Auto-TRD design?
- What hidden factors did we not capture in the Auto-TRD? Do we need to update the Auto-TRD or implement a missing feature?

Release Implemented System.

The last phase of the AASDLC is to **release** the system to users in the desired environment. By this time the overall system should meet the desired system goal and the AA goal should be discernibly met. Although design and implementation may be done, further updates may come once the system is put into practice and the practitioner should expect further iterations on different phases of the life-cycle. When the system is placed in the field, some feedback should be collected to insure that performance as expected from laboratory studies transfers to the final environment.

Comparison to Other Life-Cycles.

Since the goal of all system development life-cycles is to create an end product, this phase is similar across all life-cycles. However, what sets the AASDLC apart from the rest is that the release criteria of the end product is incumbent upon not only meeting the overall system goal, but also the underlying AA goal. If this is not met, the AA system may not work as expected.

Designer Questions.

- Does the system as implemented meet the overall system goal? Does it meet the overall AA goal?
- To which situations does the system respond well in operation? To which does it respond poorly?

- What changes could have been made to the design process to improve the transition to the real-world?
- What fundamental changes are apparent in the system that need to be communicated to the end users of the system?

4.4 Use Case

This section examines the AASDLC with an example adaptive automation system development. The authors designed and implemented AA in the *Space Navigator* [1, 138] tablet computer game. The application of the AASDLC to *Space Navigator* that follows provides an example of how the AASDLC can aid the development process, given both good and bad decisions.

Phase 1: Define Adaptive Automation Goals.

The overall goal of the *Space Navigator* system was to provide a test environment to evaluate how users interact with an adaptive automation. Specifically, the AA goal was to have the machine take over for the human in an adaptive manner and perform a portion of the overall task similarly to how the human interacting with the system would perform it. Doing this at a time when ships appeared at a rate that exceeded the human operator's ability to route the ships, monitor their paths, and re-route if necessary. The reason for performing a portion of the task similarly to a human rather than optimally was to create a system for a research project to identify the effects of similarity of action on human-machine teams. The resulting system will allow researchers to test how users respond to adaptive automations that act similarly to how they would in a given situation, in comparison to AAs that act dissimilarly.

The agent was to permit the user to safely route more spaceships to a planet than

they could on their own, thus improving their scores under high task load conditions, while permitting them to remain engaged in the game. Although the system did not exist, similar games and an early prototype had been constructed. A trajectory generation game was chosen as the environment because it provides a complex and dynamic environment, while allowing the player only one input response (drawing trajectories). The particular game design also included special characteristics including a point system that provided a unitary award (e.g. no leveling-up concept was used) and task load could be manipulated by the practitioner by changing the number of NFZs or the rate of new ship arrivals.

In summary, the overall system goal was clear: design a system to gain as many points as possible. The AA goal was relatively clear: allow the user to gain more points during periods of high task load than they could otherwise achieve, while permitting the users to remain engaged in the game. Notice this last goal has two potentially competing sub-goals. It might be possible to remove the user and obtain higher point scores but as user engagement is a key part of the requirement, such a solution is not acceptable. It is the competition of these goals which bounds the problem in a way that requires the automation to adjust its behavior in response to the task load and the user's response to this task load, making AA desirable. Further note that a more measurable goal could have been established, such as insuring that 90% of all spacecraft arrived at a planet for spacecraft spawn rates less than 1 spacecraft per second.

Phase 2: Align the design and system.

Since the system was not already built, phase two began with designing the system. The system design was performed using the FTTDPM. The resulting TRD–shown in Figure 27 was used to perform the implementation, and tweaks to the system were

captured in the TRD during implementation.

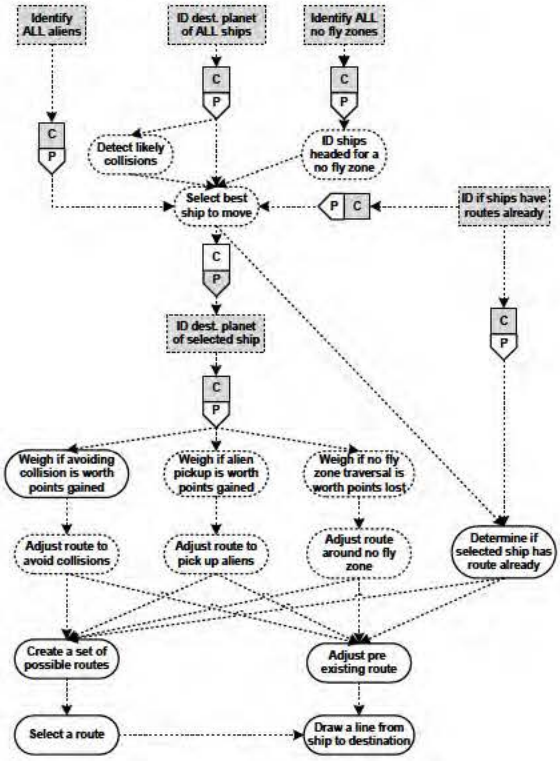

Figure 27. The *Space Navigator* TRD as captured during phase two of the AASDLC with rectangles representing machine functions, ovals representing human functions, and the C/P blocks indicating inherent tasks which occur as information is transmitted between the machine and human; adapted from [1].

One important insight gained about the system came from designing the feedback system within *Space Navigator*. In the game as originally designed, the human portion of the task is strictly selecting a ship and drawing its trajectory. However, several aspects of the game are performed by the machine: ship movement, bonus spawns, ship scoring, etc. To help the human understand the system and raise the humans' situation awareness, several implementation changes prompted an understanding of functions that needed to be added to the FRD and assigned as machine tasks in the TRD. For example, drawing waypoints for the ships' trajectories was the user's responsibility, but displaying where these points were for other ships was the responsibility of the system. By understanding what "should be communicated to

the user," the practitioner can better understand what tasks the machine is actually performing. By the end of this phase, *Space Navigator* was implemented directly from the design, there were no known problems with the alignment of the system and the implementation.

Note that in reality a significant error was originally made within Figure 27 which occurred due to the definition of the machine. In this environment, as in many AA environments, the machine potentially has many different functions. In the present example, it both generates the environment and supports the automated agent which interacts with the human and environment generator to conduct tasks that the human would have conducted. In the initial implementation, the environment generator was aware of the information shown as rectangles in Figure 27 and so these tasks were shown as machine tasks. However, from the human point of view, the intended view of these diagrams, these tasks were performed by the human as they had to perceive the representation of the ships, planets and other elements from the screen and decide upon the proper relationships. Therefore, this diagram should depict these tasks as human tasks, unless they were being delegated to the automation agent. Once this occurs, the diagram changes as shown in Figure 28. A failure to recognize this distinction at this point will be shown to affect subsequent design phases.

Phase 3: Verify design and implementation alignment through user testing.

In order to see how users interacted with the *Space Navigator* environment and to ensure design and implementation alignment, an initial user test captured a variety of data. Data was collected from 32 participants playing 16 five-minute instances of *Space Navigator* each. Although data was collected from each of the four feedback quadrants discussed in Section 4.3. Pros and cons of the actual feedback devices

implemented for the *Space Navigator* user test are discussed here.

Quantitative directed: The quantitative directed feedback sought in the experiment dealt with perceived workload. The NASA Task Load indeX (TLX) [188] and Instantaneous Self Assessment of workload (ISA) [189] batteries were presented to users after each instance. Table 4 shows the mean workload ratings reported across all users during user testing. The variations correspond to different settings in the *Space Navigator* environment with relation to spaceship spawn rates and number of NFZs. Surprisingly, the ISA indicated no change in workload as the ship spawn rate or NFZs increased but did indicate an increase in workload for the fast spawn rate/increased number of NFZ condition. Mental and temporal demand as well as frustration, as expected, generally increased with increasing spawn rate and the number of NFZs. Unfortunately, this assessment did not provide insight into the tasks which induced the most workload, which might have been more useful in designing the system. Therefore, future efforts might include using the TRD decomposition to have users rate the difficulty or importance of each sub-task.

Table 4. Mean and standard error for ISA (1-5 scale) and NASA TLX (0-100 scale) ratings as a function of new spaceship spawn rates (fast [1 *ship*/2 *seconds*] **or slow** [1 *ship*/5 *seconds*]**) and number of no-fly zones present (**2 **or** 4**) during user testing in *Space Navigator***

	Slow/ 2 NFZs	**Fast/ 2 NFZs**	**Slow/ 4 NFZs**	**Fast/ 4 NFZs**
ISA Workload	3.15 ± 0.10	3.16 ± 0.08	3.15 ± 0.09	3.29 ± 0.09
Mental Demand	50.6 ± 2.2	51.9 ± 2.0	52.0 ± 2.0	54.4 ± 2.1
Physical Demand	39.7 ± 2.3	38.2 ± 2.1	39.1 ± 2.2	40.9 ± 2.2
Temporal Demand	49.7 ± 2.2	51.2 ± 2.2	50.6 ± 2.2	53.1 ± 2.3
Frustration	40.1 ± 2.1	39.6 ± 1.9	41.3 ± 2.1	42.4 ± 2.0
Effort	48.2 ± 2.1	49.1 ± 2.0	48.4 ± 1.9	50.5 ± 2.0
Performance	63.1 ± 1.9	61.6 ± 2.0	60.6 ± 2.0	59.3 ± 1.9

Qualitative directed: Several post-test survey questions proved too vague to help ensure design and implementation alignment. However, questions involving different

types of AA that would help the user in performing the task were useful. Users provided several ideas that the system creators had not considered.

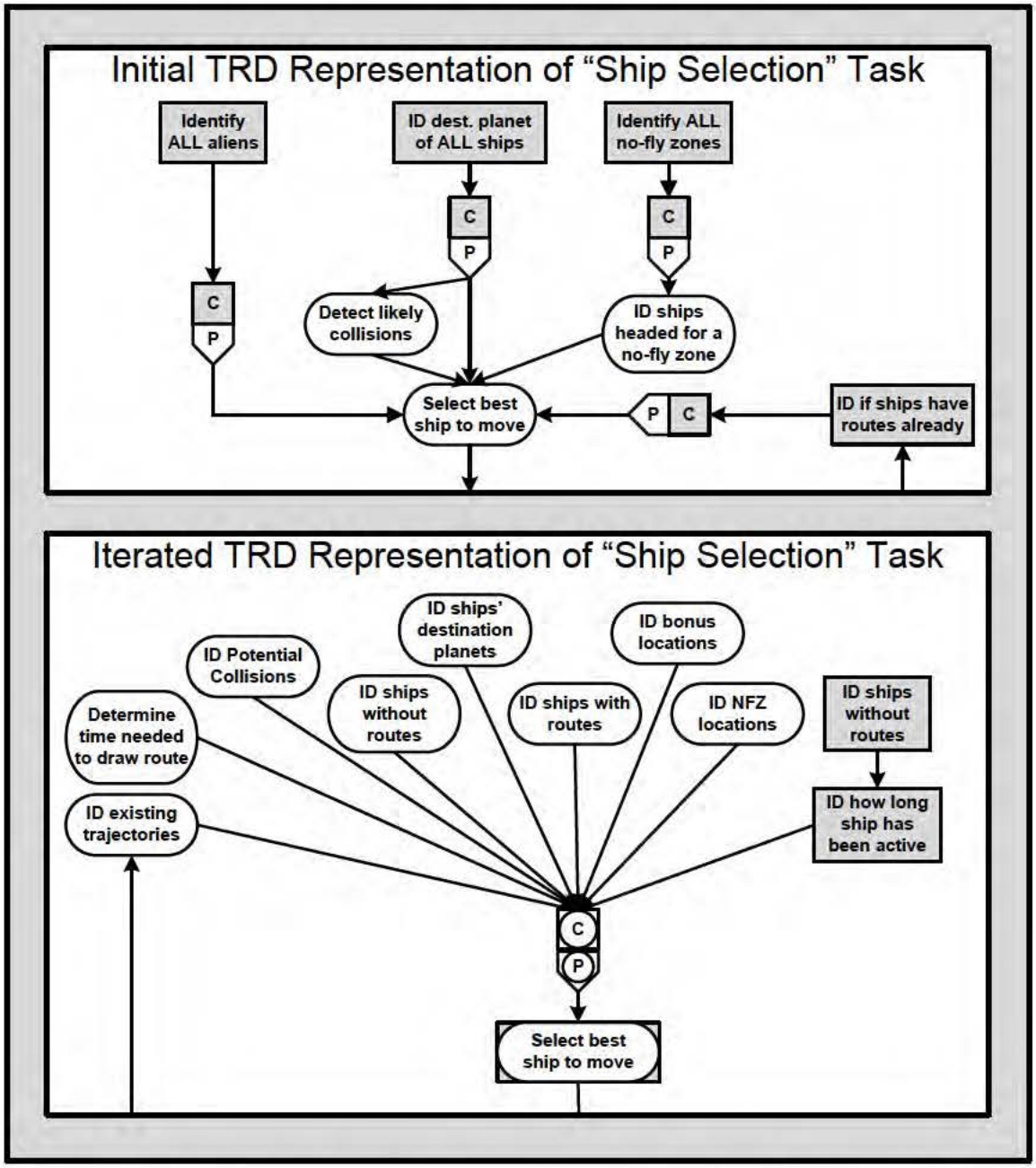

Figure 28. The 'select ship' sub-task of the *Space Navigator* TRD, comparing representation before and after the AASDLC user testing processes.

Quantitative undirected: The quantitative undirected measures proved to be the

most effective measures of the user test. For each user, every point scoring event was captured (e.g collisions, bonus pickups, etc.) and several elements of the current state were captured every time the user interacted with the system (i.e. at trajectory draws). Although having effective feedback here was useful in designing the AA device itself, it did not prove useful in helping to determine what inherent information the user was utilizing to make decisions before acting.

Qualitative undirected: The qualitative undirected measures consisted of asking the users their general thoughts about the game environment. Although this information was not as useful in specifically aligning the design and implementation it did prove useful in understanding how users described and related to the elements of the game. For example, when users would describe the bonuses within the game as 'bubbles' or 'moon rocks' it helped us to describe the system better to future users.

Phase 4: Add adaptive automation to design and implementation.

After phase three, we believed the implementation and design were aligned well and moved forward to adding AA to the system. However, the noted problems with design and implementation alignment coupled with poor AA trigger design ensured problems with the actual implementation. These problems manifested themselves in the user testing in phase five. The design decisions made at this phase are explained in the following paragraphs.

Automated Element.

Designing the automated element began with a run through the FTTDPM. As outlined in previous work [138], the trajectory draw function was chosen as the AA location. As previously explained, the overall AA goal was to have the automation perform a sub-task similarly to the human interacting with the system. The overall

AA goal then become more specific, drawing trajectories similarly to those of a specific person to improve the user's score while permitting them to remain engaged with the game.

The specific location for the AA is illustrated in Figure 27, including everything from the 'Select best ship to move' node all the way through the 'Draw a line from ship to destination' node. Although a few branches were expected to be crossed for the inputs to the ship selection process, these were dismissed as negligible as there were only five branches. However, the later TRD in Figure 28 shows that the actual number of inputs was at least nine. Choosing to automate below the 'Select best ship to move' node would have only required the crossing of one branch as opposed to the original five.

Adaptive Automation Trigger.

Once the automated element was decided, the trigger to adjust the automated element was designed. First, we needed to decide the mode of the trigger: would we be cycling between some set of automation settings (discrete), tuning some setting of the automated element (continuous) or some combination of the two. A simple trigger mode was chosen. Essentially there would be a discrete binary trigger: either the automated element would be on or off. Next, the trigger type would be decided. Part of the goal in aiding the system was to determine when an automated aid would be helpful to users. This was determined to be when the user was overwhelmed by the system.

At this point, two poor design choices were made: multiple trigger modes were inadvertently added and the trigger type did not align with a minor AA goal. The trigger was designed as a system-based trigger based on the time a specific element had been on the screen without being acted upon. This first failed the goal of creating

a trigger that was a simple binary trigger. Two criteria needed to be met in succession for the trigger to fire: (1) has the spaceship been on the screen for x seconds and (2) does the spaceship already have a trajectory. The trigger was represented poorly, as a single binary decision after its implementation. Our decision failed to account for the fundamental difference between drawing the initial trajectory for a ship and re-drawing a non-ideal trajectory.

Additionally, a minor goal of the AA had been to determine when the user was overwhelmed by the system. We assumed that this could be teased out of the system by determining if ships had been on the screen for a period of time without being acted upon (i.e. the person had not yet had a chance to act upon the spaceship due to time constraints). However, we discounted the fact that users may not act upon a ship for reasons other than being overwhelmed. A better trigger design would have either accounted for a system-only trigger type (e.g. the number of ships on the screen) or a user-based trigger (e.g. increased heart rate).

Adaptive Automation Interface.

Coupled with the poor AA trigger design decisions the misrepresentative design from Figure 28 led to a poor AA handoff. Based on the placement of the trigger, a well-designed handoff would ensure that the AA trigger would fire at a location that would result in relatively little information needing to pass, in this case, from the human to the machine. However, based on the accepted TRD model at the time and the time-based trigger, control of another sub-task was usurped: spaceship selection. Since the spaceship for which automated trajectories would be drawn was designed as a function of time, human influences into the selection decision (e.g. other ships, distance to destination, available bonuses) were ignored. The resulting similar trajectory drawing system would be up against a glass ceiling: no matter how

similarly the trajectories represented the human's response to the given state, they were doomed to be a poor representation of human trajectory draws because the user would never have SELECTED that ship in the first place.

A second, understated problem arose at the back end of the AA. When the machine gave control of the system back to the human a quirk in the AA interface design proved problematic. The automation was allowed to draw trajectories instantaneously, and new trajectories were added to the screen all at once. This allowed the system to draw trajectories at a much faster rate than the human could and also made it difficult for the human to decipher which trajectories were automated. Therefore, the individuals were often unaware that the automation had drawn a trajectory. This unexpected action decreased human trust in the system and increased the cognitive workload of the human during handoffs.

Phase 5: Verify design and implementation alignment through user testing.

To verify the alignment of the design and implementation after AA, another user test captured a variety of data. The data was collected from 35 participants playing 17 five-minute instances of *Space Navigator* each. The tests included: five initial games with no AA followed by three sets of four games cycling through four AA settings. The first setting was no automation, the second consisted of the designed similar automation. The third and fourth settings were designed as comparison measures. One automation was programmed to always draw a straight line from the ship to the destination planet and the second automation was programmed to choose a random trajectory from the past user game-play database. The final two settings were used as an experimental comparison. This design choice for the user test proved extremely beneficial in diagnosing the problems from phase four. By providing a comparison

system, the user test allowed us to identify the design issues that were due to poor AA design choices rather than problems arising from the automation's ability to perform the task.

The data collected (across all the feedback quadrants) from the 35 users allowed us to identify problems with the design. The most obvious sign of a lack of implemented AA alignment with the overall AA goal was that the designed AA system encouraged users to play the game drastically differently than they had without automation. The system as implemented allowed users to rely on the automation to draw all trajectories and engage in a trajectory correction task. Figure 29 shows the drastic change in trajectory draws that occurred as a result. Additionally, this game-play change was consistent across all AA settings, ensuring that it was not due to the trajectory generating automation mechanism, but the trigger.

The second sign of alignment problems was the directed and undirected qualitative user feedback. Users consistently identified the largest factors impacting the success of their interaction with the AA systems as the predictability of the automation and the perceived ability of the automation. Predictability and ability of the automation are well documented factors in human-machine interaction, but the fact that these factors played as heavily as they did into AA interaction suggested that the AA system did not meet the overall AA goal. Only one user of the 35 was able to discern that the system was playing in a similar manner, with several users specifically indicating that the unpredictability of the system did not allow them to understand how it was creating trajectories.

Revisiting Phases 3-5.

Based upon these results, the previous phases are revisited. An informal interview of three participants identified that the TRD did not accurately reflect the system as

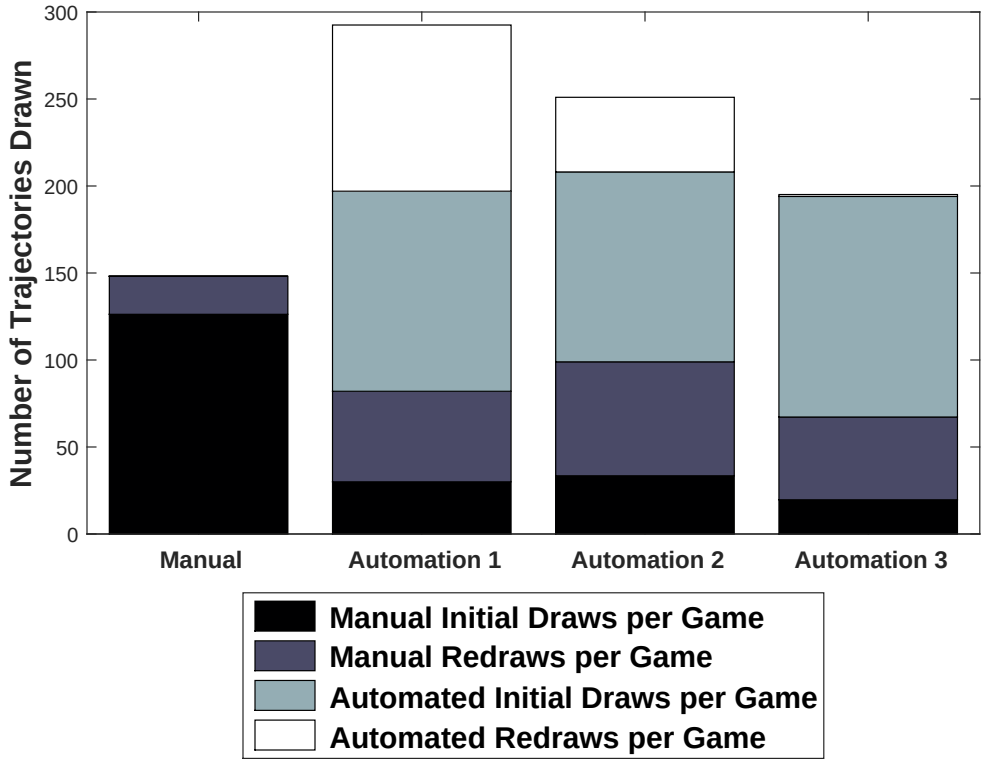

Figure 29. The number and type of trajectory draws as a function of automation type in *Space Navigator*.

designed. User feedback and game-play data were then used to redesign the TRD, identifying problems with the designed system. Two changes were made to the system with two goals: (1) ensuring the automated element within the system was better isolated from the rest of the sub-tasks within the system and (2) ensuring the machine communicated its actions more clearly to the human.

The first change was to move the location of the initial AA trigger from above the 'select best ship to move' node, as shown in Figure 28, to below this node. This allowed the interface to only require one type of input to represent all of the user's initial inputs to the selection decision, rather than the initial 10 inputs that the system must represent in making its decisions of which ships to automate.

The second change was a two-fold choice made to improve the human-computer interaction by updating the way that the machine's trajectory draws were communicated to the user. First, human and machine trajectories were shown with different colors, as opposed to only one color for all trajectories. Second, the ships trajectories were drawn to the screen one dot at a time rather than all at once, to anthropomorphize the machine's action choices in an understandable manner.

These changes were updated in the TRD and implemented into the system. An experiment comparing the previous AA with the revised AA showed that the identified changes achieved the two goals. The users surveyed stated a distinctly better automation interaction experience, citing both the improved understanding of the system due to the removal of unexpected machine actions (change 1) and the clarity gained from understanding what actions the machine was taking and had taken (change 2). Additionally, in a head-to-head comparison of the two automation settings (before and after changes), users unanimously chose the improved automation as more similar to the way they would have drawn trajectories than the previous automation system.

Phase 6: Release implemented System.

The release of the implemented system is still ongoing. As the process continues, further insights are developed and the phases of the process reiterate accordingly, leading to a stronger system implementation and better understanding of how the design represents the reality of this implementation. Iterating through the changes throughout the process has shown the usefulness of a cyclical process and the clear need for design and implementation refinement throughout the process.

4.5 Conclusions

The Adaptive Automation System Development Life-Cycle, a beginning-to-end system development life-cycle for designing and implementing an adaptive automation system. The AASDLC allows for significant flexibility in its actual execution, as most systems will not follow a linear path from conception to system actualization. The UCD focuses of the life-cycle enable the practitioner to ensure that the system can account for these deviations by allowing user feedback to ensure design and implementation align throughout the process. Additionally, separating the process into pre- and post- AA sections allows the practitioner to apprise the gains provided by AA within the system.

Iterating through the AASDLC's phases of alignment and verification through user testing produces an implemented system that works as designed and a system design that reflects the reality of that system. These products give designers a better understanding of the operational system, enabling them to effectively communicate the systems capabilities to stakeholders and train users on the implemented system. Iterating through the process creates a better understanding of how adding AA to the system can enable a better human-machine system to meet the overall system's goals.

V. Conclusions

This dissertation presented a path for system designers to take practical adaptive automation (AA) goals and move them into a real-world AA system. Specifically, we have addressed the question "How do we design and implement a real-world adaptive automation system around a specific adaptive automation goal?"

The answers to this question provide three contributions to the research community. The Function to Task Design Process Model (FTTDPM) for AA system design, Chapter II and [1], aids system designers in defining AA goals and locating areas in a system that would benefit from AA. A real-world AA system developed as the automation portion of the system, Chapter III and [87, 138], gives researchers an individual player modeling method that updates in real-time. The AA development work all ties together under the Adaptive Automation System Design Life Cycle (AASDLC), Chapter IV and [81], establishing a process for taking the AA designs derived from application of the FTTDPM and transferring them into real-world systems.

The FTTDPM encompasses seven steps that form an iterative AA system design process. Moving from determining over-arching system goals to defining AA states within a codified system design, FTTDPM is a non-linear process that emphasizes the importance of task instantiation within a system design. Several, often unforeseen, inherent tasks result from assigning the operation of a specific function to a specific human or machine entity. The FTTDPM accounts for these inherent tasks through a set of three diagrams that additively represent a robust AA system: the Function Relationship Diagrams (FRD), Task Relationship Diagrams (TRD), and TRD with automation added (Auto-TRD). Using these diagrams in conjunction with a set of five new analysis metrics tied to the TRD framework (number of possible states, number of different entity task handoffs, clusters of functionality, number of branches

and increase in inherent task load), system designers can design AA solutions that fit well within a specific system.

Implementing an AA system in the *Space Navigator* automation environment, which was created to learn how to move from an AA system design to a real-world system, required the creation of a novel real-time individual player modeling system to generate trajectories within the environment. To meet the designed AA goal of performing a sub-task of the *Space Navigator* environment similarly to how the human in the loop would have done it required creating a trajectory generator that would imitate the trajectories a specific person would generate in response to a given state. The solution to this problem was a player modeling technique involving three major phases that can occur independently of each other or may overlap in a real-time system: (1) create a generic player model, (2) update the individual player model, and (3) generate a response using the player model. The process provided three key contributions to the player modeling and game artificial intelligence communities: the player model updates in real-time, it learns player tendency quickly, and it provides practitioners valuable insights into how the player interacts with the environment.

The AA system design and *Space Navigator* automation from previous steps then formed the backbone of a real-world AA system implementation that influenced the development of the AASDLC, a six-phase start-to-finish development life-cycle. The AASDLC addresses implementation concerns specific to AA systems by dividing the system implementation into pre- and post-AA portions. By assuring that the design and system align at every phase, more effective use of the FTTDPM principles can help to produce real-world systems that align with user expectations. In conjunction with the major contribution of the life-cycle itself, the AASDLC contributes three in three ways to the adaptive automation and system design communities: (1) incorporating the AA-centered design principles of the FTTDPM, (2) creating a new user

feedback spectrum, and (3) developing a novel model for AA triggers.

5.1 Discussion

Each of the major contributions provided insights that warrant further discussion. The FTTDPM research raises considerations on function decomposition and distinguishing between the environment and the automation acting within it. The player modeling research raises questions about the differences between modeling novice and experts within a system. The AASDLC user testing uncovered the potential importance of automation predictability and complementarity as major design considerations. Additionally, choices made when designing the *Space Navigator* environment affected experimental outcomes.

The Function-to-Task Design Process Model.

One of the problems that can arise with the FTTDPM comes from function decomposition not being universal. Although the functions performed by a human-machine system can be decomposed systematically in several ways, there is no way to ensure with absolute certainty that the functionalities represented within the resulting FRD are representative of the actual system. As such, design decisions can be made on faulty representations of the system. For this reason, understanding the goals of both the automation and the operator within the system is important. Additionally, different people may organize functionality relationships in different ways. The isolation of functionality is important to the FTTDPM and as such, can cause problems with systems that are representable along multiple hierarchical lines. These function decomposition problems can cause a problem in determining when to end the function decomposition step of the process model.

Another discussion point raised during the AASDLC is the disentanglement of the

automated element of the AA system and the environment in which the automated element operates. In many human-machine system environments, the environment will operate within the same platform as the machine element of the system. For example, in *Space Navigator* the environment that generates spaceships, collects points, and displays events operates in the same computer program as several elements of the automated element. Within the context of a radar monitoring environment, an automated aid could be a computer program that helps the user detect anomalies that would be hard to disentangle from the radar monitoring environment. This can become complicated specifically at the point of human-machine handoffs. Thus, it is important to clearly define where the environment begins and the automation system ends.

Clustering-Based Real-Time Player Modeling.

The player modeling tactic chosen was made to learn a specific users playing habits very quickly, but this mode of learning has positive and negative effects. It was beneficial to learn a player's habits in the first five games to quickly move from a generic player model to a specific player model that outperformed that generic model. The problem, however, came from users also learning how to perform better in the system over time. By training on only five games, the player models were learning player tendencies that may not be carried forward as the user becomes an expert in the environment. The player model had to not only pull out player-specific game-play patterns, but also avoid modeling those behavior patterns that would not carry over into future game-play.

In this sense, understanding the difference between novices, normal skilled, and expert practitioners of a system would help in creating effective player models. A few of the hallmarks of expert skill include consistency [190–192], automatization [80,190,

193–196], anomaly sensitivity [80, 192, 194], global decision-making [195, 196], and a large time investment [191, 192, 194, 195]. In a generic sense, an expert in a given field is someone who performs consistently well, can do so intuitively, can notice slight changes in the environment, can take the larger environment into account during the decision process, and has devoted significant time and effort into the acquisition of specific skill. By taking these items into account, a better understanding of what character traits should be modeled at what point in time–and how these change over time could prove useful in player modeling at an early stage.

An Adaptive Automation System Development Life-Cycle.

A lack of alignment between the design and system proved problematic in the third set of user tests, specifically introducing unpredictability which proved more troublesome than expected. By not ensuring that the system design represented the system as implemented before doing our third user data collection experiment, we ensured that the experiment would not reap the expected results. When the first version of the AA trajectory drawing system used the time-based trigger, users were confused and bristled at the system's unpredictability. Initial results in the experiment, therefore, pointed to a possible problem. At this point it would have been beneficial to stop the user testing, analyze the data we had on hand, and redesign the AA triggers for the system. Even though the designed system was more similar to the users' way of drawing trajectories than the other automations used, users could not get past the fact that the automation was not understandable. Although this finding follows with those of past research efforts [197], the extent to which unpredictability dominated user attention and understanding of the automation was beyond expectation.

A few users also mentioned a vague concept of play style complementarity in au-

tomation. The users said that they liked the style of automation that most complemented their personal playing style. One user who cited a desire for complementary automation, noted that the straight-line automation in the third user-test complemented his style of getting the ships moving toward their destination quickly. While another user said that having a similar playing system could potentially complement his actions if he knew that the automation would be similar to what he would do. Although there is no definitive proof, it seems that the ideas of complementary system design in [198] could provide insight into designing effective automated elements.

Research Domain Considerations.

The *Space Navigator* domain was created as a domain complex enough to hinder automation's ability to perform "optimally" in the environment, while being intuitive to users. The domain was made complex through a set of dynamically changing state elements and a diverse set of event-triggering scoring mechanisms. By limiting users' responses to these changing states to one action, the domain became intuitive. However, the intuitive nature of the environment may have enabled some unforeseen consequences.

The state-space is dynamic. No-fly zones move randomly around the screen at set intervals, so users cannot predict where a NFZ will show up next. Bonuses appear randomly, and remain in place until they are picked up. Most importantly, new spaceships are spawned in random locations. These dynamic aspects of the environment make it difficult for the automation to create a "best possible" route at any given point in time. Based on the known information, the best possible route for a given ship may change drastically when NFZs move or a new ship appears.

There are multiple types of scoring mechanisms. Although all scoring events provide either a positive or negative score increment, each scoring event type impacts

the environment differently. As currently designed, when a spaceship lands at its intended target, it scores a one-time 100 point increment, and the ship disappears from the state-space–it "lands." Bonuses give a 50 point bonus, but do not affect the makeup of the environment in any way–the ship just continues on its course. Collisions remove 100 points per ship involved in a crash, and remove at least two ships from the environment–which can have a positive "decluttering" effect. No-fly zones affect the environment similarly to bonuses–ships just continue their paths–but the scoring mechanism is a repeated 10 point penalty until the ship leaves the NFZ.

Because of its simple interaction mechanism–the lone user input method is to draw trajectories–*Space Navigator* is intuitive to users. Users may only take one type of action, but process a diverse set of inputs to make their input decisions. Because the input is so simple and intuitive, users have a hard time explaining why they made specific decisions. User surveys provided insight into low-level tactics (e.g. sending ships off the screen, creating ingress lanes, etc.), but expert users had great difficulty in differentiating strategies that set their game-play in higher scoring instances in comparison to lower scoring instances.

The task representations shown in the TRD helped to show that users, although they were only performing "one action," were actually making a set of complex decisions all manifested in a trajectory draw. Additionally, the simplistic nature of the user response proved difficult to represent fully in the TRD. Since users do not necessarily think sequentially when making decisions in the trajectory draw process, the TRD representation does have limits to its representation abilities. This becomes especially important when choosing how to represent an environment state representation scheme.

5.2 Future Work

This research leaves many doors open for future efforts and collaborations. These possible directions include but are not limited to:

- *Further using player modeling to compare game-play tendencies of players before and after automation changes are added* - This could include using the existing player modeling techniques or even modifying the techniques, such as self-organizing maps instead of agglomerative clustering, to model player tendencies across a broad spectrum of aspects of the user response (e.g. trajectory draw time, draw start times, etc.) in addition to the trajectories themselves. These player models could then be used to compare game-play tendencies before and after automations are added to a system. Comparing the models could provide insight into the different ways that disparate types of automation affect users within the environment.

- *Comparison of different automation types for training users* - A new user test could be designed where each user is only exposed to one type of automation over an extended period of time to determine which automations are helpful in training users to perform well in a given environment (e.g. *Space Navigator*). A pre- and post experiment baseline would be taken to determine the user's game-play capabilities before and after training. Then the user would train on a specific automation type or none at all for a set number of repetitions. Observations of user performance increases for the different automations could prove useful to AA system design researchers.

- *Improving the specific player game-play similarity automated element* - An obvious next step for the trajectory generation portion of the research would be to improve the ability of the trajectory generator to produce similar trajectories.

Improvements could be sought in state representation, ability to learn a player model quickly, or even creating a better generic baseline to build from. Better result measuring of action similarity could also improve the ability of the system to measure how well the system imitates even more aspects of the user's tendencies.

- *Re-running the user-test with improved automation to tease out effects of similar game-play* - Additionally, once the final improvements of the *Space Navigator* environment are implemented, another user test could be run trying to tease out the similarity aspect of the automation. Controlling for problems that were present in the initial design will help to remove some of the distractions users experienced. It could also be useful to ensure that the comparative systems were all equally adept in the environment on their own and also similarly predictable in nature. A good comparison may come from having the system imitate the user in question, and then imitate another very different user.

- *Analysis of subjective workload measures in comparison to real-world measures of competence in the environment* - We have collected subjective data from users over two experiments with the ISA and NASA TLX workload batteries. We could look into the data to determine what the data says about views of workload, and whether they line up with objective measures of performance such as score.

- *Control theory for AA trigger design* - Further theoretical work could be done on the design of adaptive automation triggers. There is evidence within the AASDLC that seems to point toward a lack of understanding of what adaptability truly entails within the AA community. It is likely that all AA system triggers could be represented by a set of discrete and continuous switches. Ap-

plying control theory research to AA trigger design could open up synergies within the two research fields.

Bibliography

1. J. M. Bindewald, M. E. Miller, and G. L. Peterson, "A function-to-task process model for adaptive automation system design," *International Journal of Human-Computer Studies*, vol. 72, no. 12, pp. 822–834, 2014.

2. H. E. Price, "The allocation of functions in systems," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 27, no. 1, pp. 33–45, 1985.

3. K. M. Feigh, M. C. Dorneich, and C. C. Hayes, "Toward a characterization of adaptive systems a framework for researchers and system designers," *Human Factors*, vol. 54, no. 6, pp. 1008–1024, 2012.

4. R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 30, no. 3, pp. 286–297, 2000.

5. T. B. Sheridan and R. Parasuraman, "Human-automation interaction," *Reviews of human factors and ergonomics*, vol. 1, no. 1, pp. 89–129, 2005.

6. Merriam-Webster, "Adapt," 2013. [Online]. Available: http://www.merriam-webster.com/dictionary/adapt

7. W. B. Rouse, "Human-computer interaction in multitask situations," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 5, pp. 384–392, 1977.

8. ——, "Human-computer interaction in the control of dynamic systems," *ACM Computing Surveys (CSUR)*, vol. 13, no. 1, pp. 71–99, 1981.

9. M. C. Dorneich, P. M. Ververs, S. Mathan, S. Whitlow, and C. C. Hayes, "Considering etiquette in the design of an adaptive system," *Journal of Cognitive Engineering and Decision Making*, vol. 6, no. 2, pp. 243–265, 2012.

10. R. Parasuraman, "Supporting battle management command and control: Designing innovative interfaces and selecting skilled operators," DTIC Document, Tech. Rep., 2008.

11. A. Fereidunian, M. Lehtonen, H. Lesani, C. Lucas, and M. Nordman, "Adaptive autonomy: smart cooperative cybernetic systems for more humane automation solutions," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*. IEEE, 2007, pp. 202–207.

12. G. R. J. Hockey, D. G. Wastell, and J. Sauer, "Effects of sleep deprivation and user interface on complex performance: a multilevel analysis of compensatory control," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 40, no. 2, pp. 233–253, 1998.

13. J. Sauer, D. Wastell, and G. Hockey, "A conceptual framework for designing micro-worlds for complex work domains: a case study of the cabin air management system," *Computers in Human Behavior*, vol. 16, no. 1, pp. 45–58, 2000.

14. B. Lorenz, F. Di Nocera, S. Röttger, and R. Parasuraman, "Automated fault-management in a simulated spaceflight micro-world," *Aviation, Space, and Environmental Medicine*, vol. 73, no. 9, pp. 886–897, 2002.

15. J. Sauer, C.-S. Kao, D. Wastell, and P. Nickel, "Explicit control of adaptive automation under different levels of environmental stress," *Ergonomics*, vol. 54, no. 8, pp. 755–766, 2011.

16. M. Mahfouf, J. Zhang, D. A. Linkens, A. Nassef, P. Nickel, G. R. J. Hockey, and A. C. Roberts, "Adaptive fuzzy approaches to modelling operator functional states in a human-machine process control system," 2007.

17. C.-H. Ting, M. Mahfouf, D. A. Linkens, A. Nassef, P. Nickel, A. C. Roberts, M. H. Roberts, and G. R. J. Hockey, "Real-time adaptive automation for performance enhancement of operators in a human-machine system," in *Proceedings of the 16th Mediterranean Conference on Control and Automation*. IEEE, 2008, pp. 552–557.

18. C.-H. Ting, M. Mahfouf, D. A. Linkens, A. Nassef, P. Nickel, G. R. J. Hockey, and A. C. Roberts, "Towards on-line supervision and control of operational functional state (ofs) for subjects under mental stress," in *Proceedings of the 2007 IEEE/NIH Life Science Sysytems and Applications Workshop (LISSA 2007)*. IEEE, 2008, pp. 77–80.

19. C.-H. Ting, M. Mahfouf, A. Nassef, D. A. Linkens, G. Panoutsos, P. Nickel, A. C. Roberts, and G. Hockey, "Real-time adaptive automation system based on identification of operator functional state in simulated process control operations," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 40, no. 2, pp. 251–262, 2010.

20. J. Sauer, C.-S. Kao, and D. Wastell, "A comparison of adaptive and adaptable automation under different levels of environmental stress," *Ergonomics*, vol. 55, no. 8, pp. 840–853, 2012.

21. R. Johnson, M. Leen, and D. Goldberg, "Testing adaptive levels of automation (aloa) for uav supervisory control," DTIC Document, Tech. Rep., 2007.

22. G. L. Calhoun, H. A. Ruff, M. H. Draper, and E. J. Wright, "Automation-level transference effects in simulated multiple unmanned aerial vehicle control," *Journal of Cognitive Engineering and Decision Making*, vol. 5, no. 1, pp. 55–82, 2011.

23. G. L. Calhoun, V. B. Ward, and H. A. Ruff, "Performance-based adaptive automation for supervisory control," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 55, no. 1. SAGE Publications, 2011, pp. 2059–2063.

24. G. L. Calhoun, H. A. Ruff, S. Spriggs, and C. Murray, "Tailored performance-based adaptive levels of automation," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 56, no. 1. SAGE Publications, 2012, pp. 413–417.

25. B. Kidwell, G. L. Calhoun, H. A. Ruff, and R. Parasuraman, "Adaptable and adaptive automation for supervisory control of multiple autonomous vehicles," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 56. SAGE Publications, 2012, pp. 428–432.

26. P. M. Ververs, S. Whitlow, M. Dorneich, and S. Mathan, "Building honeywells adaptive system for the augmented cognition program," in *1st International Conference on Augmented Cognition, Las Vegas, NV*, 2005.

27. M. C. Dorneich, P. M. Ververs, S. D. Whitlow, S. Mathan, J. Carciofini, and T. Reusser, "Neuro-physiologically-driven adaptive automation to improve decision making under stress," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 50, no. 3. Sage Publications, 2006, pp. 410–414.

28. J. Comstock and R. Arnegard, "Multi-attribute task battery (nasa tech. memorandum 104174)," *Hampton, VA: NASA Langley Research Center*, 1992.

29. R. Parasuraman, M. Mouloua, and R. Molloy, "Effects of adaptive task allocation on monitoring of automated systems," *Human Factors*, vol. 38, no. 4, pp. 665–679, 1996.

30. R. Parasuraman, M. Mouloua, and B. Hilburn, "Adaptive aiding and adaptive task allocation enhance human-machine interaction," *Automation technology and human performance: Current research and trends*, pp. 119–123, 1999.

31. F. G. Freeman, P. J. Mikulka, L. J. Prinzel, and M. W. Scerbo, "Evaluation of an adaptive automation system using three eeg indices with a visual tracking task," *Biological Psychology*, vol. 50, no. 1, pp. 61–76, 1999.

32. N. R. Bailey, M. W. Scerbo, F. G. Freeman, P. J. Mikulka, and L. A. Scott, "Comparison of a brain-based adaptive system and a manual adaptable system for invoking automation," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 48, no. 4, pp. 693–709, 2006.

33. M. C. Dorneich, B. Passinger, C. Hamblin, C. Keinrath, J. Vašek, S. D. Whitlow, and M. Beekhuyzen, "The crew workload manager an open-loop adaptive system design for next generation flight decks," in *Proceedings of the Human Factors*

and Ergonomics Society Annual Meeting, vol. 55, no. 1. SAGE Publications, 2011, pp. 16–20.

34. D. Barber, L. Davis, D. Nicholson, N. Finkelstein, and J. Y. Chen, "The mixed initiative experimental (mix) testbed for human robot interactions with varied levels of automation," DTIC Document, Tech. Rep., 2008.

35. C. M. Fidopiastis, J. Drexler, D. Barber, K. Cosenzo, M. Barnes, J. Y. Chen, and D. Nicholson, "Impact of automation and task load on unmanned system operators eye movement patterns," in *Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience*. Springer, 2009, pp. 229–238.

36. K. Cosenzo, J. Chen, L. Reinerman-Jones, M. Barnes, and D. Nicholson, "Adaptive automation effects on operator performance during a reconnaissance mission with an unmanned ground vehicle," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 54, no. 25. SAGE Publications, 2010, pp. 2135–2139.

37. D. B. Kaber and M. R. Endsley, "Out-of-the-loop performance problems and the use of intermediate levels of automation for improved control system functioning and safety," *Process Safety Progress*, vol. 16, no. 3, pp. 126–131, 1997.

38. D. B. Kaber and J. M. Riley, "Adaptive automation of a dynamic control task based on secondary task workload measurement," *International journal of cognitive ergonomics*, vol. 3, no. 3, pp. 169–187, 1999.

39. M. P. Clamann, M. C. Wright, and D. B. Kaber, "Comparison of performance effects of adaptive automation applied to various stages of human-machine system information processing," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 46, no. 3. SAGE Publications, 2002, pp. 342–346.

40. M. P. Clamann and D. B. Kaber, "Authority in adaptive automation applied to various stages of human-machine system information processing," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 47, no. 3. SAGE Publications, 2003, pp. 543–547.

41. D. B. Kaber, J. M. Riley, K.-W. Tan, and M. R. Endsley, "On the design of adaptive automation for complex systems," *International Journal of Cognitive Ergonomics*, vol. 5, no. 1, pp. 37–57, 2001.

42. D. B. Kaber and M. R. Endsley, "The effects of level of automation and adaptive automation on human performance, situation awareness and workload in a dynamic control task," *Theoretical Issues in Ergonomics Science*, vol. 5, no. 2, pp. 113–153, 2004.

43. D. B. Kaber, M. C. Wright, L. J. Prinzel, and M. P. Clamann, "Adaptive automation of human-machine system information-processing functions," *Human*

Factors: The Journal of the Human Factors and Ergonomics Society, vol. 47, no. 4, pp. 730–741, 2005.

44. C. F. Rusnock and C. D. Geiger, "The impact of adaptive automation invoking thresholds on cognitive workload and situational awareness," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1. SAGE Publications, 2013, pp. 119–123.

45. ——, "Using discrete-event simulation for cognitive workload modeling and system evaluation," in *Proceedings of the 2013 Industrial and Systems Engineering Research Conference*, A. Krishnamurthy and W. Chan, Eds., 2013.

46. M. Omodei, G. Elliott, and M. Walshe, "Development of computer simulated wildfire scenarios for the experimental investigation of unsafe decision making," Brushfire Cooperative Research Center, Tech. Rep. 2:2004, June 2006.

47. R. S. Gutzwiller and B. A. Clegg, "Training for unmanned vehicle allocation with automation in a dynamic microworld," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 56, no. 1. SAGE Publications, 2012, pp. 2497–2501.

48. R. S. Gutzwiller, B. A. Clegg, C. Smith, J. E. Lewis, and J. D. Patterson, "Predicted failure alerting in a supervisory control task does not always enhance performance," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1. SAGE Publications, 2013, pp. 364–368.

49. C. A. Cook, C. Corbridge, C. A. Morgan, and A. J. Tattersall, "Developing dynamic function allocation for future naval systems," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 41, no. 2. SAGE Publications, 1997, pp. 1047–1051.

50. C. Cook, C. Corbridge, C. Morgan, and E. Turpin, "Investigating methods of dynamic function allocation for naval command and control," in *Human Interfaces in Control Rooms, Cockpits and Command Centres, 1999. International Conference on*. IET, 1999, pp. 388–393.

51. C. Morgan, C. Cook, and C. Corbridge, "Dynamic function allocation for naval command and control," *Automation technology and human performance: Current research and trends*, pp. 134–138, 1999.

52. Y. Boussemart and M. Cummings, "Behavioral recognition and prediction of an operator supervising multiple heterogeneous unmanned vehicles," *Humans operating unmanned systems*, 2008.

53. M. Cummings and C. Nehme, "Modeling the impact of workload in network centric supervisory control settings," in *2nd Annual Sustaining Performance Under Stress Symposium,(College Park, MD)*, 2009.

54. M. Cummings, S. Bruni, S. Mercier, and P. Mitchell, "Automation architecture for single operator, multiple uav command and control," DTIC Document, Tech. Rep., 2007.

55. D. Gartenberg, L. A. Breslow, J. Park, J. M. McCurry, and J. G. Trafton, "Adaptive automation and cue invocation: the effect of cue timing on operator error," in *Proceedings of the 2013 ACM annual conference on Human factors in computing systems*. ACM, 2013, pp. 3121–3130.

56. D. Gartenberg, L. Breslow, J. M. McCurry, and J. G. Trafton, "Situation awareness recovery," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, pp. 1–18, 2013.

57. K. A. Cosenzo, R. Parasuraman, A. Novak, and M. Barnes, "Implementation of automation for control of robotic systems," DTIC Document, Tech. Rep., 2006.

58. R. Parasuraman, M. Barnes, K. Cosenzo, and S. Mulgund, "Adaptive automation for human-robot teaming in future command and control systems," DTIC Document, Tech. Rep., 2007.

59. K. Cosenzo, R. Parasuraman, K. Pillalamarri, and T. Feng, "The effect of appropriately and inappropriately applied automation for the control of unmanned systems on operator performance," DTIC Document, Tech. Rep., 2009.

60. K. Cosenzo, R. Parasuraman, and K. Pillalamarri, "The effect of task based automation for the control of unmanned systems on operator performance," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 4. SAGE Publications, 2008, pp. 247–251.

61. R. Parasuraman, K. A. Cosenzo, and E. De Visser, "Adaptive automation for human supervision of multiple uninhabited vehicles: Effects on change detection, situation awareness, and mental workload," *Military Psychology*, vol. 21, no. 2, pp. 270–297, 2009.

62. T. Inagaki, "Situation-adaptive autonomy: Trading control of authority in human-machine systems," *Automation technology and human performance: Current research and trends*, pp. 154–159, 1999.

63. N. Moray, T. Inagaki, and M. Itoh, "Adaptive automation, trust, and self-confidence in fault management of time-critical tasks." *Journal of Experimental Psychology: Applied*, vol. 6, no. 1, p. 44, 2000.

64. M. Gacy and D. Dahn, "Commonality of control paradigms for unmanned systems," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 339–340.

65. E. de Visser and R. Parasuraman, "Adaptive aiding of human-robot teaming effects of imperfect automation on performance, trust, and workload," *Journal of Cognitive Engineering and Decision Making*, vol. 5, no. 2, pp. 209–231, 2011.

66. G. F. Wilson and C. A. Russell, "Psychophysiologically determined adaptive aiding in a simulated ucav task," *Human performance, situation awareness, and automation: Current research and trends*, pp. 200–204, 2004.

67. ——, "Performance enhancement in an uninhabited air vehicle task using psychophysiologically determined adaptive aiding," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 49, no. 6, pp. 1005–1018, 2007.

68. A. Valente, E. Carpanzano, A. Nassehi, and S. T. Newman, "A step compliant knowledge based schema to support shop-floor adaptive automation in dynamic manufacturing environments," *CIRP Annals-Manufacturing Technology*, vol. 59, no. 1, pp. 441–444, 2010.

69. A. Valente and E. Carpanzano, "Development of multi-level adaptive control and scheduling solutions for shop-floor automation in reconfigurable manufacturing systems," *CIRP Annals-Manufacturing Technology*, vol. 60, no. 1, pp. 449–452, 2011.

70. D. B. Kaber, E. Onal, and M. R. Endsley, "Design of automation for telerobots and the effect on performance, operator situation awareness, and subjective workload," *Human Factors and Ergonomics in Manufacturing*, vol. 10, no. 4, pp. 409–430, 2000.

71. D. B. Kaber, M. C. Wright, and M. A. Sheik-Nainar, "Investigation of multimodal interface features for adaptive automation of a human–robot system," *International journal of human-computer studies*, vol. 64, no. 6, pp. 527–540, 2006.

72. R. Murphy and J. Shields, "The role of autonomy in dod systems," Technical report, Department of Defense, Defense Science Board Task Force Report, Tech. Rep., 2012.

73. J. A. Winnefeld, Jr. and F. Kendall, "Unmanned systems integrated roadmap fy2013-2038," Department of Defense, Tech. Rep. 14–S–0553, 2013.

74. J. Eggers and M. H. Draper, "Multi-uav control for tactical reconnaissance and close air support missions: Operator perspectives and design challenges," in *Proc. NATO RTO Human Factors and Medicine Symp. HFM-135. NATO TRO, Neuilly-sur-Siene, CEDEX, Biarritz, France*, 2006.

75. Office of the US Air Force Chief Scientist, "Technology horizons: A vision for air force science and technology 2010-30," Office of the US Air Force Chief Scientist, Tech. Rep. AF/ST-TR-10-01-PR, September 2011.

76. M. R. Endsley, "Level of automation effects on performance, situation awareness and workload in a dynamic control task," *Ergonomics*, vol. 42, no. 3, pp. 462–492, 1999.

77. R. Parasuraman, "Designing automation for human use: empirical studies and quantitative models," *Ergonomics*, vol. 43, no. 7, pp. 931–951, 2000.

78. E. de Visser, M. S. Cohen, M. LeGoullon, O. Sert, A. Freedy, E. Freedy, G. Weltman, and R. Parasuraman, "A design methodology for controlling, monitoring, and allocating unmanned vehicles," in *3rd International Conference on Human Centered Processes. Delft: Proceeding of Supervisory Control in Critical Systems Management Workshop*, 2008.

79. A. Haarmann, W. Boucsein, and F. Schaefer, "Combining electrodermal responses and cardiovascular measures for probing adaptive automation during simulated flight," *Applied Ergonomics*, vol. 40, no. 6, pp. 1026–1040, 2009.

80. M. W. Wiggins, "The role of cue utilisation and adaptive interface design in the management of skilled performance in operations control," *Theoretical Issues in Ergonomics Science*, no. ahead-of-print, pp. 1–10, 2012.

81. J. M. Bindewald, G. L. Peterson, M. E. Miller, and B. T. Langhals, "An adaptive automation system design life cycle," *IEEE Transactions on Human Machine Systems*, SUBMITTED.

82. W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, no. 8. Los Angeles, 1970.

83. B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

84. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The kdd process for extracting useful knowledge from volumes of data," *Communications of the ACM*, vol. 39, no. 11, pp. 27–34, 1996.

85. R. Wirth and J. Hipp, "Crisp-dm: Towards a standard process model for data mining," in *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*. Citeseer, 2000, pp. 29–39.

86. G. Di Orio, J. Barata, C. Sousa, and L. Flores, "Control system software design methodology for automotive industry," in *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3848–3853.

87. J. M. Bindewald, G. L. Peterson, and M. E. Miller, "Clustering-based real-time player modeling," *IEEE Transactions on Computational Intelligence and AI in Games*, SUBMITTED.

88. M. Itoh, G. Abe, and K. Tanaka, "Trust in and use of automation: their dependence on occurrence patterns of malfunctions," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3. IEEE, 1999, pp. 715–720.

89. T. Prevot, J. Homola, and J. Mercer, "Human-in-the-loop evaluation of ground-based automated separation assurance for nextgen," in *Congress of International Council of the Aeronautical Sciences Anchorage, Anchorage, AK*, 2008.

90. R. Parasuraman and C. D. Wickens, "Humans: Still vital after all these years of automation," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 50, no. 3, pp. 511–520, 2008.

91. H. Nakazawa, "Alternative human role in manufacturing," *AI & society*, vol. 7, no. 2, pp. 151–156, 1993.

92. D. D. Woods and R. I. Cook, "Incidents–markers of resilience or brittleness," *Resilience Engineering. Concepts and Precepts. Ashgate, Aldershot, UK*, 2006.

93. M. Itoh, "A model of trust in automation: Why humans over-trust," in *SICE Annual Conference (SICE), 2011 Proceedings of.* IEEE, 2011, pp. 198–201.

94. M. T. Dzindolet, S. A. Peterson, R. A. Pomranky, L. G. Pierce, and H. P. Beck, "The role of trust in automation reliance," *International Journal of Human-Computer Studies*, vol. 58, no. 6, pp. 697–718, 2003.

95. J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 46, no. 1, pp. 50–80, 2004.

96. S. M. Merritt, H. Heimbaugh, J. LaChapell, and D. Lee, "I trust it, but i don?t know why effects of implicit attitudes toward automation on trust in an automated system," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 2012.

97. B. Kirwan, "Developing human informed automation in air traffic management," *Contemporary Issues In Human Factors And Aviation Safety*, p. 247, 2005.

98. A. Bye, E. Hollnagel, and T. S. Brendeford, "Human–machine function allocation: a functional modelling approach," *Reliability Engineering & System Safety*, vol. 64, no. 2, pp. 291–300, 1999.

99. W. MacDonald, "The impact of job demands and workload on stress and fatigue," *Australian Psychologist*, vol. 38, no. 2, pp. 102–117, 2003.

100. G. B. Reid and T. Nygren, "The subjective workload assessment technique: A scaling procedure for measuring mental workload," *Human mental workload*, vol. 185, p. 218, 1988.

101. P. Fitts, A. Chapanis, F. Frick, W. Garner, J. Gebhard, W. Grether, R. Henneman, W. Kappauf, E. Newman, and A. Williams, Jr., "Human engineering for an effective air-navigation and traffic-control system," National Research Council Committe on Aviation Psychology, Washington, D.C., Tech. Rep., 1951.

102. J. A. B. Calleja and J. Troost, "A fuzzy expert system for task distribution in teams under unbalanced workload conditions," in *International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, vol. 1. IEEE, 2005, pp. 549–556.

103. T. B. Sheridan and W. L. Verplank, "Human and computer control of undersea teleoperators," DTIC Document, Tech. Rep., 1978.

104. E. E. Geiselman, C. M. Johnson, and D. R. Buck, "Flight deck automation invaluable collaborator or insidious enabler?" *Ergonomics in Design: The Quarterly of Human Factors Applications*, vol. 21, no. 3, pp. 22–26, 2013.

105. E. A. Byrne and R. Parasuraman, "Psychophysiology and adaptive automation," *Biological Psychology*, vol. 42, no. 3, pp. 249–268, 1996.

106. C. D. Wickens, "Multiple resources and mental workload," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 50, no. 3, pp. 449–455, 2008.

107. D. Manzey, J. Reichenbach, and L. Onnasch, "Human performance consequences of automated decision aids the impact of degree of automation and system experience," *Journal of Cognitive Engineering and Decision Making*, vol. 6, no. 1, pp. 57–87, 2012.

108. R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "Situation awareness, mental workload, and trust in automation: Viable, empirically supported cognitive engineering constructs," *Journal of Cognitive Engineering and Decision Making*, vol. 2, no. 2, pp. 140–160, 2008.

109. D. A. Norman and S. W. Draper, *User Centered System Design; New Perspectives on Human-Computer Interaction.* Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1986.

110. J. Gulliksen, B. Göransson, I. Boivie, S. Blomkvist, J. Persson, and Å. Cajander, "Key principles for user-centred systems design," *Behaviour and Information Technology*, vol. 22, no. 6, pp. 397–409, 2003.

111. D. Ross, "Structured analysis (sa): A language for communicating ideas," *Software Engineering, IEEE Transactions on*, vol. SE-3, no. 1, pp. 16–34, Jan 1977.

112. L. Delligatti, *SysML Distilled: A Brief Guide To The Systems Modeling Language.* Addison-Wesley, 2013.

113. C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson Education, Inc, 2004.

114. J. Annett and K. D. Duncan, "Task analysis and training design." 1967.

115. J. Annett, "Hierarchical task analysis," *Handbook of cognitive task design*, pp. 17–35, 2003.

116. B. S. Blanchard and W. J. Fabrycky, *Systems engineering and analysis*, 2006.

117. M. Van Welie and G. C. Van Der Veer, "Groupware task analysis," *Handbook of cognitive task design*, pp. 447–476, 2003.

118. P. Johnson, H. Johnson, R. Waddington, and A. Shouls, "Task-related knowledge structures: analysis, modelling and application," in *BCS HCI*. Citeseer, 1988, pp. 35–62.

119. F. Paterno, "Towards a uml for interactive systems," in *Engineering for human-computer interaction*. Springer, 2001, pp. 7–18.

120. S. Lu, C. Paris, K. V. Linden, and N. Colineau, "Generating uml diagrams from task models," in *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction*. ACM, 2003, pp. 9–14.

121. S. K. Card, T. P. Moran, and A. Newell, *The psychology of human-computer interaction*. CRC Press, 1986.

122. J.-C. Tarby and M.-F. Barthet, "The diane+ method." in *CADUI*, vol. 96, 1996, pp. 95–119.

123. J. Vanderdonckt, J.-C. Tarby, and A. Derycke, "Using data flow diagrams for supporting task models." in *DSV-IS (2)*, 1998, pp. 1–16.

124. F. Paternò, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models," in *Human-Computer Interaction INTERACT97*. Springer, 1997, pp. 362–369.

125. M. Giese, T. Mistrzyk, A. Pfau, G. Szwillus, and M. von Detten, "Amboss: A task modeling approach for safety-critical systems," in *Engineering Interactive Systems*. Springer, 2008, pp. 98–109.

126. E. Hollnagel, *Fram: the functional resonance analysis method: modelling complex socio-technical systems*. Ashgate Publishing, Ltd., 2012.

127. W. van der Aalst and A. ter Hofstede, "Yawl: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245 – 275, 2005.

128. D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. Knig, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu, Eds., *Web Services Business Process Execution Language Version 2.0*, 2nd ed., April 2007.

129. C. Martinie, P. Palanque, E. Barboni, and M. Ragosta, "Task-model based assessment of automation levels: application to space ground segments," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3267–3273.

130. C. Martinie, P. Palanque, M. Ragosta, and R. Fahssi, "Extending procedural task models by systematic explicit integration of objects, knowledge and information," in *Proceedings of the 31st European Conference on Cognitive Ergonomics*. ACM, 2013, p. 23.

131. D. M. Buede, *The engineering design of systems: models and methods*. John Wiley & Sons, 2011, vol. 55.

132. A. Cheng-Leong, K. Li Pheng, and G. R. Keng Leng, "Idef*: a comprehensive modelling methodology for the development of manufacturing enterprise systems," *International Journal of Production Research*, vol. 37, no. 17, pp. 3839–3858, 1999.

133. J. Schrum, I. V. Karpov, and R. Miikkulainen, "Human-like combat behaviour via multiobjective neuroevolution," in *Believable bots*. Springer, 2012, pp. 119–150.

134. D. Gamez, Z. Fountas, and A. K. Fidjeland, "A neurally controlled computer game avatar with humanlike behavior," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 1, pp. 1–14, 2013.

135. M. Kemmerling, N. Ackermann, and M. Preuss, "Making diplomacy bots individual," in *Believable Bots*. Springer, 2012, pp. 265–288.

136. H. Yu and M. O. Riedl, "Personalized interactive narratives via sequential recommendation of plot points," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 2, pp. 174–187, 2014.

137. R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations: a survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 2, pp. 85–99, 2011.

138. J. M. Bindewald, G. L. Peterson, and M. E. Miller, "Trajectory generation with player modeling," in *Advances in Artificial Intelligence*. Springer, 2015, pp. 42–49.

139. G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player modeling," *Artificial and Computational Intelligence in Games*, vol. 6, pp. 45–59, 2013.

140. D. Charles and M. Black, "Dynamic player modeling: A framework for player-centered digital games," in *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004, pp. 29–35.

141. C. Bateman, R. Lowenhaupt, and L. E. Nacke, "Player typology in theory and practice," in *Proceedings of DiGRA*, 2011.

142. B. Weber and M. Mateas, "A data mining approach to strategy prediction," in *IEEE CIG 2009*, Sept 2009, pp. 140–147.

143. J. Gow, S. Colton, P. A. Cairns, and P. Miller, "Mining rules from player experience and activity data." in *AIIDE*, 2012.

144. A. Drachen, A. Canossa, and G. Yannakakis, "Player modeling using self-organization in tomb raider: Underworld," in *IEEE CIG 2009*, Sept 2009, pp. 1–8.

145. J. Gow, R. Baumgarten, P. Cairns, S. Colton, and P. Miller, "Unsupervised modeling of player style with lda," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 3, pp. 152–166, Sept 2012.

146. C.-X. Zhang, Z.-K. Zhang, L. Yu, C. Liu, H. Liu, and X.-Y. Yan, "Information filtering via collaborative user clustering modeling," *Physica A: Statistical Mechanics and its Applications*, vol. 396, no. 0, pp. 195 – 203, 2014.

147. A. M. Smith, C. Lewis, K. Hullet, and A. Sullivan, "An inclusive view of player modeling," in *Proceedings of the 6th International Conference on Foundations of Digital Games*. ACM, 2011, pp. 301–303.

148. J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," *Optimizing Player Satisfaction in Computer and Physical Games*, p. 61, 2006.

149. S. C. Bakkes, P. H. Spronck, and G. van Lankveld, "Player behavioural modelling for video games," *Entertainment Computing*, vol. 3, no. 3, pp. 71–79, 2012.

150. J. Rubin and I. Watson, "Computer poker: A review," *Artificial Intelligence*, vol. 175, no. 56, pp. 958 – 987, 2011, special Review Issue. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370211000191

151. K. Laviers, G. Sukthankar, D. W. Aha, M. Molineaux, C. Darken *et al.*, "Improving offensive performance through opponent modeling." in *AIIDE*, 2009.

152. S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Case-Based Reasoning Research and Development*. Springer, 2007, pp. 164–178.

153. S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in starcraft," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.

154. G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 2, pp. 121–133, 2009.

155. G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Affective Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 147–161, 2011.

156. S. Begum, M. U. Ahmed, P. Funk, N. Xiong, and M. Folke, "Case-based reasoning systems in the health sciences: A survey of recent trends and developments," *Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, vol. 41, no. 4, pp. 421–434, July 2011.

157. B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

158. T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

159. C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, 1999.

160. B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. ACM, 2007, pp. 57–64.

161. M. W. Floyd, B. Esfandiari, and K. Lam, "A case-based reasoning approach to imitating robocup players," in *FLAIRS Conference*, 2008, pp. 251–256.

162. S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Learning from demonstration and case-based planning for real-time strategy games," in *Soft Computing Applications in Industry*. Springer, 2008, pp. 293–310.

163. M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, 2009.

164. J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

165. H. H. S. T. Huang, Victor and C. J. Tomlin, "Contrails: Crowd-sourced learning of human models in an aircraft landing game," in *Proceedings of the AIAA GNC Conference*, 2013.

166. B. Morris and M. Trivedi, "Learning trajectory patterns by clustering: Experimental studies and comparative evaluation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 312–319.

167. X. Li, W. Hu, and W. Hu, "A coarse-to-fine strategy for vehicle motion trajectory clustering," in *ICPR 2006*, vol. 1. IEEE, 2006, pp. 591–594.

168. D. M. Lane, Ed., *Introduction to Statistics: An interactive eBook*. Rice University, 2013.

169. M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *ICRA 2006*. IEEE, 2006, pp. 3344–3349.

170. J. Wiest, M. Hoffken, U. Kresel, and K. Dietmayer, "Probabilistic trajectory prediction with gaussian mixture models," in *IEEE Intelligent Vehicles Symposium*. IEEE, 2012, pp. 141–146.

171. C. Miller, M. E. Miller, G. L. Calhoun *et al.*, "Triggering changes in adaptive automation evaluation of task performance, priority and frequency," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1732–1737.

172. T. Bhuvaneswari and S. Prabaharan, "A survey on software development life cycle models," *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 5, pp. 262–267, May 2013.

173. D. Green and A. DiCaterino, *A survey of system development process models*. Center for Technology in Government, University at Albany-SUNY, 1998.

174. K. Schwaber, "Scrum development process," in *Business Object Design and Implementation*, J. Sutherland, C. Casanave, J. Miller, P. Patel, and G. Hollowell, Eds. Springer London, 1997, pp. 117–134.

175. C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

176. K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

177. J. Martin, *Rapid application development*. Macmillan publishing company, 1991.

178. A. Azevedo and M. F. Santos, "Kdd, semma and crisp-dm: A parallel overview," in *Proceedings of the IADIS European Conference on Data Mining*, July 2008, pp. 182–185.

179. H. A. Edelstein, "Introduction to data mining and knowledge discovery," 1998.

180. G. Mariscal, Ó. Marbán, and C. Fernández, "A survey of data mining and knowledge discovery process models and methodologies," *The Knowledge Engineering Review*, vol. 25, no. 02, pp. 137–166, 2010.

181. S. Ahangama and D. C. C. Poo, *International Journal of Medical, Health, Biomedical and Pharmaceutical Engineering*, vol. 8, no. 11, pp. 768–776, 2014. [Online]. Available: http://waset.org/Publications?p=95

182. S. Benbelkacem, M. Belhocine, N. Zenati-Henda, A. Bellarbi, and M. Tadjine, "Integrating human-computer interaction and business practices for mixed reality systems design: a case study," *Software, IET*, vol. 8, no. 2, pp. 86–101, 2014.

183. K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey, "A survey of user-centered design practice," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2002, pp. 471–478.

184. M. R. Endsley, *Designing for situation awareness: An approach to user-centered design*. CRC Press, 2011.

185. I. Newman and C. R. Benz, *Qualitative-quantitative research methodology: Exploring the interactive continuum*. SIU Press, 1998.

186. T. Inagaki, "Adaptive automation: Sharing and trading of control," *Handbook of cognitive task design*, vol. 8, pp. 147–169, 2003.

187. C. Miller, H. Funk, P. Wu, R. Goldman, J. Meisner, and M. Chapman, "The playbook approach to adaptive automation," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 49, no. 1. SAGE Publications, 2005, pp. 15–19.

188. S. G. Hart and L. E. Staveland, "Development of NASA-TLX (task load index): Results of empirical and theoretical research," *Advances in psychology*, vol. 52, pp. 139–183, 1988.

189. B. Kirwan, A. Evans, L. Donohoe, A. Kilner, T. Lamoureux, T. Atkinson, and H. MacKendrick, "Human factors in the atm system design life cycle," in *proceedings of EUROCONTROL/FAA ATM R & D Seminar*, 1997, pp. 16–20.

190. A. Renkl and R. K. Atkinson, "Structuring the transition from example study to problem solving in cognitive skill acquisition: A cognitive load perspective," *Educational psychologist*, vol. 38, no. 1, pp. 15–22, 2003.

191. K. A. Ericsson and A. C. Lehmann, "Expert and exceptional performance: Evidence of maximal adaptation to task constraints," *Annual review of psychology*, vol. 47, no. 1, pp. 273–305, 1996.

192. J. Shanteau, D. J. Weiss, R. P. Thomas, J. Pounds, and B. Hall, "How can you tell if someone is an expert? empirical assessment of expertise," *Emerging perspectives on judgement and decision research*, pp. 620–639, 2003.

193. D. L. Mann, B. Abernethy, and D. Farrow, "Action specificity increases anticipatory performance and the expert advantage in natural interceptive tasks," *Acta Psychologica*, vol. 135, no. 1, pp. 17–23, 2010.

194. A. Didierjean and F. Gobet, "Sherlock holmes–an expert's view of expertise," *British Journal of Psychology*, vol. 99, no. 1, pp. 109–125, 2008.

195. M. Harré, T. Bossomaier, and A. Snyder, "The development of human expertise in a complex environment," *Minds and Machines*, vol. 21, no. 3, pp. 449–464, 2011.

196. R. W. Proctor and T. Van Zandt, *Human factors in simple and complex systems*. Boca Raton: CRC Press, 2008, ch. Experts and Expert Systems, pp. 315–337.

197. S. C. Sutherland, C. Harteveld, and M. E. Young, "The role of environmental predictability and costs in relying on automation," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 2535–2544. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702609

198. G. Grote, C. Ryser, T. WĀLER, A. Windischer, and S. Weik, "Kompass: A method for complementary function allocation in automated work systems," *International Journal of Human-Computer Studies*, vol. 52, no. 2, pp. 267–287, 2000.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
09–17–2015	Doctoral Dissertation	Sept 2012 — Sep 2015

4. TITLE AND SUBTITLE

Adaptive Automation Design and Implementation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Bindewald, Jason M., Capt, USAF

5d. PROJECT NUMBER

13RSL-AFIT1

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-DS-15-S-007

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

US Air Force Office of Scientific Research, Cognition and Decision
Attn: James H. Lawton, PhD
4075 Wilson Blvd., Suite 350
Arlington, VA 22203
(703)696-5999 (DSN: 426-5999) James.Lawton.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material has been declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Automations allow us to reduce the need for humans in certain environments, such as auto-pilot features on unmanned aerial vehicles. However, some situations still require human intervention. Adaptive automation is a research field that enables computer systems to adjust the amount of automation by taking over tasks from or giving tasks back to the user. This research develops processes and insights for adaptive automation designers to take theoretical adaptive automation ideas and develop them into real-world adaptive automation system. These allow developers to design better automation systems that recognize the limits of computers systems, enabling better designs for systems in fields such as multi-aircraft control. This research was sponsored by AFOSR.

15. SUBJECT TERMS

Adaptive Automation, System Design, Human-Computer Interaction, Function Allocation, Player Modeling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson, AFIT/ENG
U	U	U	U	168	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4281; gilbert.peterson@afit.edu